

第 編

高度情報化支援ソフトウェアシーズ育成事業

「オブジェクト指向ソフトウェア向け
リファクタリングツールの開発」

操 作 説 明 書

- 目 次 -

| | | |
|-------|---|----|
| 1 . | 開発ソフトウェアの機能一覧 | 1 |
| 1.1 | プログラム編集機能 | 2 |
| 1.2 | リファクタリング機能 | 2 |
| 1.3 | JRB の機能構成 | 3 |
| 2 . | 開発ソフトウェアの起動・終了方法 | 4 |
| 2.1 | JRB の起動 | 4 |
| 2.2 | JRB の終了 | 5 |
| 3 . | 開発ソフトウェアの操作方法 | 6 |
| 3.1 | 基本画面 | 6 |
| 3.2 | メニュー | 7 |
| 3.2.1 | File メニュー (ファイルの基本操作) | 7 |
| 3.2.2 | Edit メニュー (ソースコードの編集機能) | 9 |
| 3.2.3 | Refactoring メニュー (リファクタリングの実行) | 11 |
| 3.2.4 | Options メニュー (オプション設定) | 12 |
| 3.2.5 | Help メニュー (ヘルプ) | 14 |
| 3.3 | 補足 | 15 |
| 4 . | リファクタリング操作 | 17 |
| 4.1 | CLASS メニュー (クラスに関するリファクタリング) | 18 |
| 4.1.1 | クラス名の変更(Rename Class) | 19 |
| 4.1.2 | クラスの移動(Move Class) | 21 |
| 4.1.3 | クラスの合併(Merge Class) | 23 |
| 4.1.4 | クラスの削除>Delete Class) | 25 |
| 4.1.5 | サブクラスの抽出(Extract Subclass) | 26 |
| 4.1.6 | スーパークラスの抽出(Extract Superclass) | 27 |
| 4.1.7 | スーパーインタフェースの抽出(Extract Super Interface) | 28 |
| 4.1.8 | インタフェースの抽出(Extract Interface) | 29 |
| 4.2 | METHOD メニュー (メソッドに関するリファクタリング) | 30 |

| | | |
|-------|--|----|
| 4.2.1 | メソッド名の変更(Rename Method)..... | 31 |
| 4.2.2 | メソッドの移動(Move Method) | 33 |
| 4.2.3 | メソッドの削除>Delete Method)..... | 35 |
| 4.2.4 | メソッドの引き上げ(Pull Up Method)..... | 36 |
| 4.2.5 | メソッドの引き下げ(Push Down Method) | 38 |
| 4.3 | FIELD メニュー (フィールドに関するリファクタリング) | 40 |
| 4.3.1 | フィールド名の変更(Rename Field) | 41 |
| 4.3.2 | フィールドの移動(Move Field)..... | 43 |
| 4.3.3 | フィールドの削除>Delete Field) | 46 |
| 4.3.4 | フィールドの引き上げ(Pull Up Field) | 47 |
| 4.3.5 | フィールドの引き下げ(Push Down Field)..... | 48 |
| 4.3.6 | カプセル化(Encapsulate Field)..... | 49 |
| 4.3.6 | 自己カプセル化(Self Encapsulate Field)..... | 51 |
| 4.4 | VARIABLE メニュー (ローカル変数に関するリファクタリング) | 53 |
| 4.4.1 | 変数名の変更(Rename Variable)..... | 54 |
| 4.4.2 | 変数の削除>Delete Variable)..... | 55 |
| 4.4.3 | スライスの抽出(Slice on Variable) | 56 |
| 4.5 | MISCELLANEOUS メニュー (その他のリファクタリング) | 57 |
| 4.5.1 | polymorphism による条件分岐(switch 文)の置き換え(Switch to Polymorphism) .. | 58 |
| 4.6 | HISTORY メニュー (リファクタリング履歴の表示) | 61 |
| 4.7 | GUIDE メニュー (リファクタリング履歴の検索および次の操作の提案) | 63 |
| 4.8 | UNDO メニュー (リファクタリング操作の取り消し) | 64 |
| 5. | 開発ソフトウェアの障害対処方法 | 65 |

1. 開発ソフトウェアの機能一覧

本ソフトウェアは、Java 言語で記述されたオブジェクト指向プログラムに対して、プログラマの行うリファクタリング操作を自動化することで、その作業を支援するツールである。以下、このソフトウェアを JRB (Java Refactoring Browser)と呼ぶ。リファクタリングとは、ソフトウェアの外部の振る舞いを保存したままで、そのソースコードを変換し、内部のソフトウェア構造を改善することを指す。図 1 に開発ソフトウェアの利用イメージを示す。

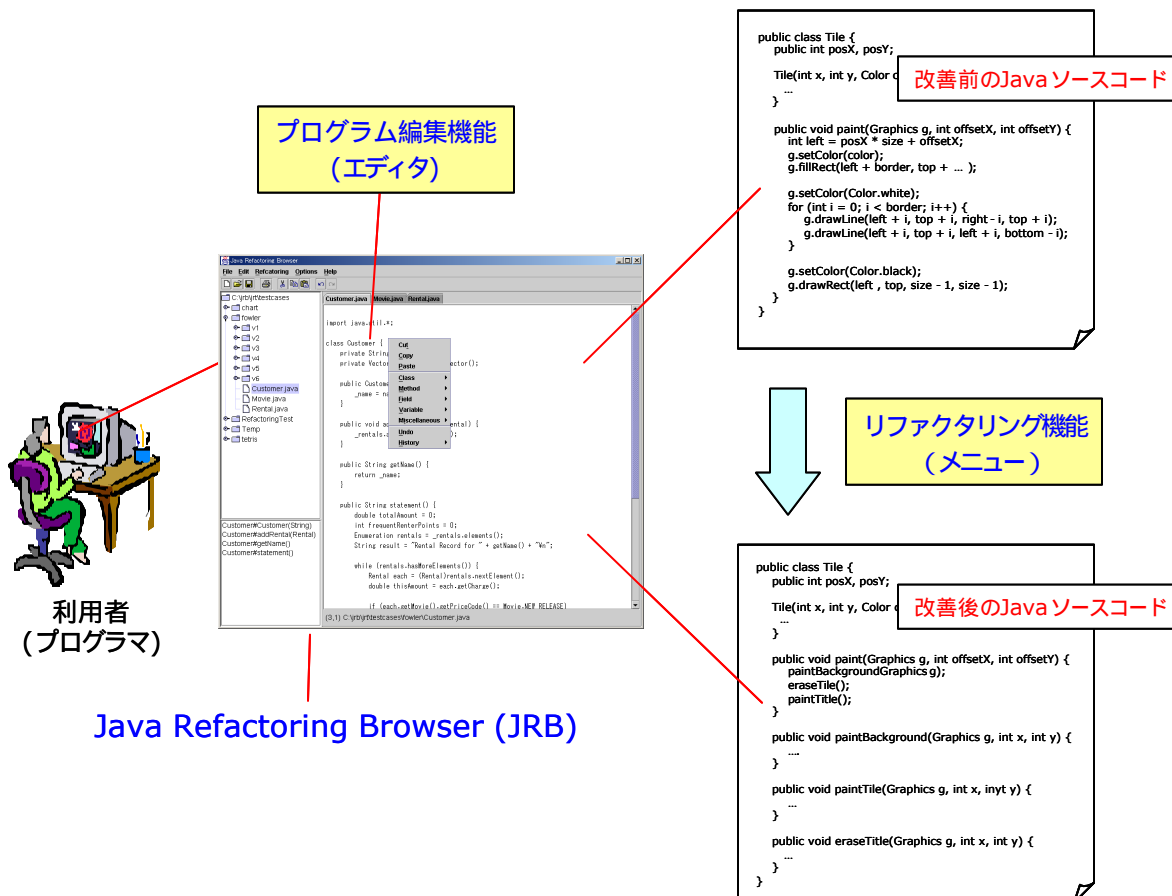


図 1 開発ソフトウェアの利用イメージ

JRB は、Java プログラム開発に対して、以下に示す 2 つの機能を提供する。

- (1) プログラム編集機能
- (2) リファクタリング機能

利用者は、JRB によって提供されるエディタ上でソースコードを編集する。リファクタリングを行う場合は、ソースコードの対象部分をエディタ上で指定し、メニューからリファクタリング操作を選択する。それぞれの機能について、1.1 および 1.2 に示す。

1.1 プログラム編集機能

Java ソースコードを編集するためのエディタおよびファイル操作を提供する．編集に関する機能を以下に示す．

- ソースコード（ファイル）の新規作成，読み込み，保存，名前変更，印刷機能
- ソースコード（テキスト）の切り取り，コピー，貼り付け，挿入，削除機能
- ソースコード上での文字列の検索，置換機能
- ソースコードに対する編集操作の取り消し，やり直し機能

1.2 リファクタリング機能

利用者が編集中のソースコードに対して，GUI を介して入力したリファクタリング操作を適用し，変換後のソースコードを編集画面に出力する．JRB では，リファクタリング適用前後の Java ソースコードの外部的振る舞いを保存するために，プログラム解析（字句解析，構文解析，フロー解析，依存解析，スライス抽出）に基づきリファクタリング操作を実行する．リファクタリングに関する機能を以下に示す．

- プログラム解析（字句解析，構文解析，依存解析，スライス抽出）に基づく各種リファクタリング操作の適用機能．JRB の提供するリファクタリング操作は次のように分類される．
 - （１）クラスに関するリファクタリング
 - （２）メソッドに関するリファクタリング
 - （３）フィールドに関するリファクタリング
 - （４）ローカル変数に関するリファクタリング
 - （５）switch 文に関するリファクタリング
- メニューによるリファクタリング操作の指定機能
- 利用者が過去に適用したリファクタリング操作履歴を蓄積，表示する機能
- 利用者の行った直前（１あるいは２つ前）のリファクタリング操作に応じて，過去の操作履歴から同一の操作を検索することで，次の操作を提案する機能
- リファクタリング操作のやり直し機能

1.3 JRB の機能構成

本説明書（1.1 および 1.2）における各種機能と，JRB システムの機能構成（「論文」および「ソフトウェア設計書」における機能構成）との対応を表 1 に示す．

表 1 JRB の機能構成

| システムの機能構成 | 本説明書における機能 |
|-------------|---|
| ユーザインタフェース部 | ファイル新規作成，読み込み，保存，名前変更，印刷 テキストの切り取り，コピー，貼り付け，挿入，削除 文字列の検索，置換 編集操作の取り消し，やり直し |
| 構文解析部 | 対象ソースコードの字句解析，構文解析 |
| 依存解析部 | 対象ソースコードの依存解析，スライスの抽出 |
| コード変換部 | 各種リファクタリング操作 リファクタリングの取り消し |
| リファクタリング操作部 | メニューによるリファクタリング操作の指定 |
| 操作列検索部 | リファクタリング操作履歴の蓄積，表示 リファクタリング操作の検索，次操作の提案 |

2 . 開発ソフトウェアの起動・終了方法

2.1 JRB の起動

(1) Windows2000/98/Me の場合

インストールフォルダにおいて , JRB.BAT アイコン  をダブルクリックする .

(2) Solaris8(UNIX)の場合

インストールディレクトリにおいて , シェルスクリプト jrb を起動する .

% ./jrb ↵

JRB は , 起動時に以下に示すプロパティファイルから環境変数を読み込む .

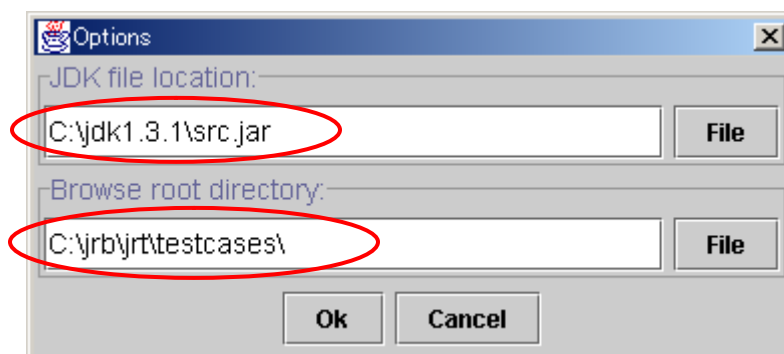
(1) JRB.properties.Windows (Windows の場合)

```
#Java Refactoring Browser properties for Windows 2000
#Fri Jan 04 15:42:03 JST 2002
Root.Dir=C¥:¥¥jp¥¥ac¥¥ritsumei¥¥cs¥¥fse¥¥jrt¥¥
JDK.File=C¥:¥¥jdk1.3.1¥¥src.jar
Application.Dir=C¥:¥¥jrb
```

(2) JRB.properties.Unix (Solaris の場合)

```
#Java Refactoring Browser properties for SunOS
#Fri Jan 04 15:45:03 JST 2002
Root.Dir=/home/jp/ac/ritsumei/cs/fse/jrt
JDK.File=/usr/local/j2sdk1_3_1/src.jar
Application.Dir=/home/maru/jrb
```

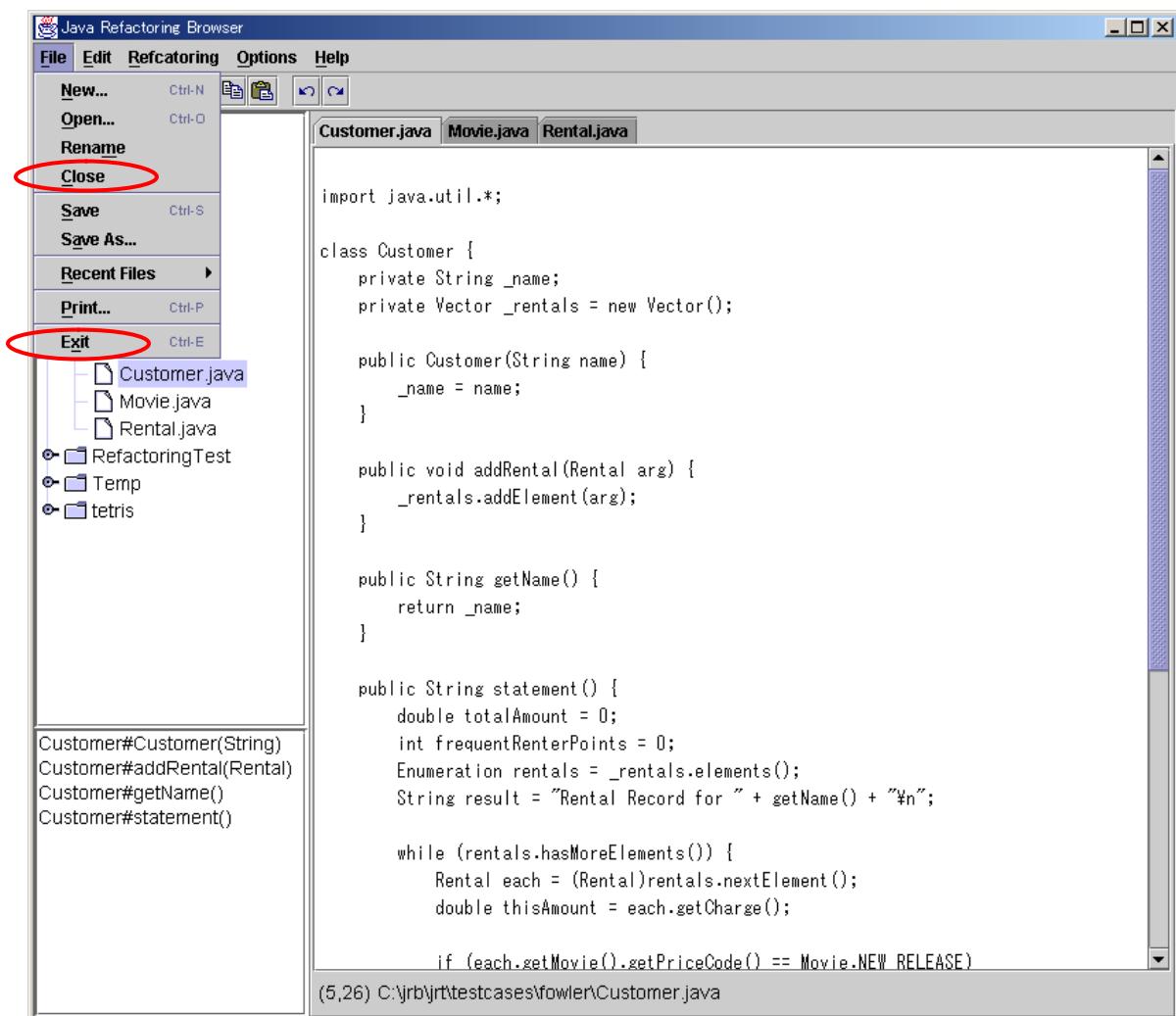
環境変数 Root.Dir は , プログラマが開発中の (リファクタリング対象とする) ソースコード群が格納されているトップディレクトリ (フォルダ) を指す . 以下 , 説明において , Root directory により指定されているディレクトリを探索ディレクトリと呼ぶ . 環境変数 JDK.File は JDK のソースコードのアーカイブファイル (jar あるいは zip 形式) を指す . Root.Dir により指定されたディレクトリ , あるいは , JDK.File により指定されたファイルが存在しない場合 , これらの環境変数の入力を促すダイアログが現れるので , それぞれの値を入力し , Ok をクリックする . この時点で Cancel をクリックすると , JRB は起動しない .



2.2 JRB の終了

JRB は、次の 2 通りのどちらかの方法で終了できる。

- (a) JRB の File メニューから Exit を選択する。
- (b) JRB の File メニューから Close を選択することで、すべてのファイルを閉じる。

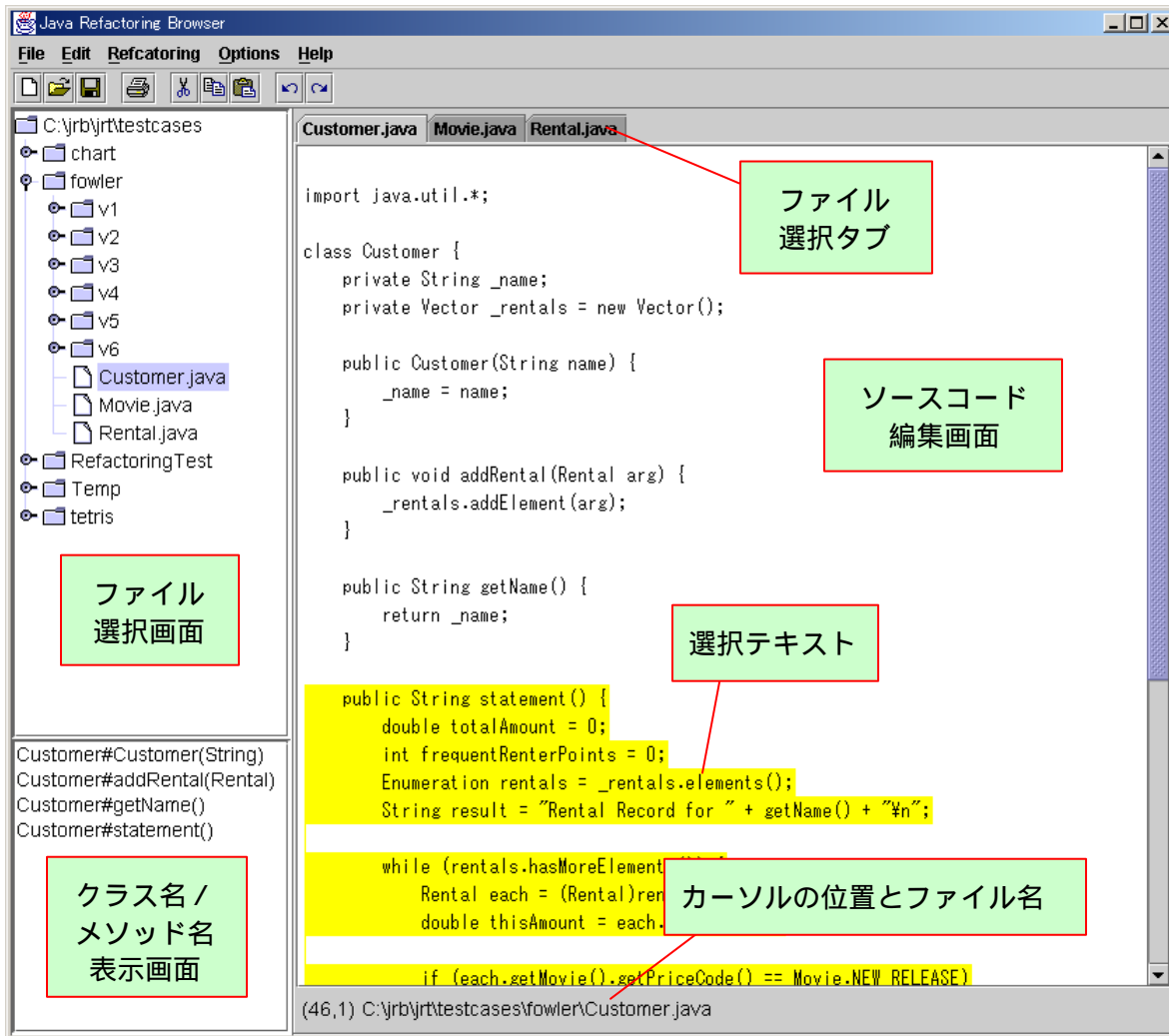


OS やウィンドウマネジャーなどにより JRB を強制終了させた場合、未保存のファイルにおける変更や環境変数の変更は破棄されるので注意すること。

3 . 開発ソフトウェアの操作方法

3.1 基本画面

JRBの基本画面を以下に示す．基本画面は，ファイル選択画面，ソースコード編集画面，クラス名／メソッド名表示画面の3つに分けられている．



(1) ファイル選択画面

探索ディレクトリ(環境変数 `Root.Dir` で指定されたディレクトリ)以下の Java ソースコード(拡張子が `java` である)ファイルがディレクトリの階層に従い表示される．この画面上でファイル名をクリックすると，指定されたファイルがファイル編集画面に読み込まれる．すでに読み込み済みのファイルの場合は，指定ファイルの編集画面が最前面に呼び出される．

(2) ソースコード編集画面

利用者に指定されたファイルの内容(ソースコード)が表示される．複数のソースコードが読

み込まれている場合、それぞれのソースコードのタブをクリックすることで、指定ソースコードを最前面に呼び出すことができる。利用者は、この画面内でマウスをドラッグすることにより対象テキストを選択し(実際のソフトウェアでは編集画面上で黄色表示になる)、ソースコードの編集、検索、置換、リファクタリング操作を行う。

(3) クラス名/メソッド名表示画面

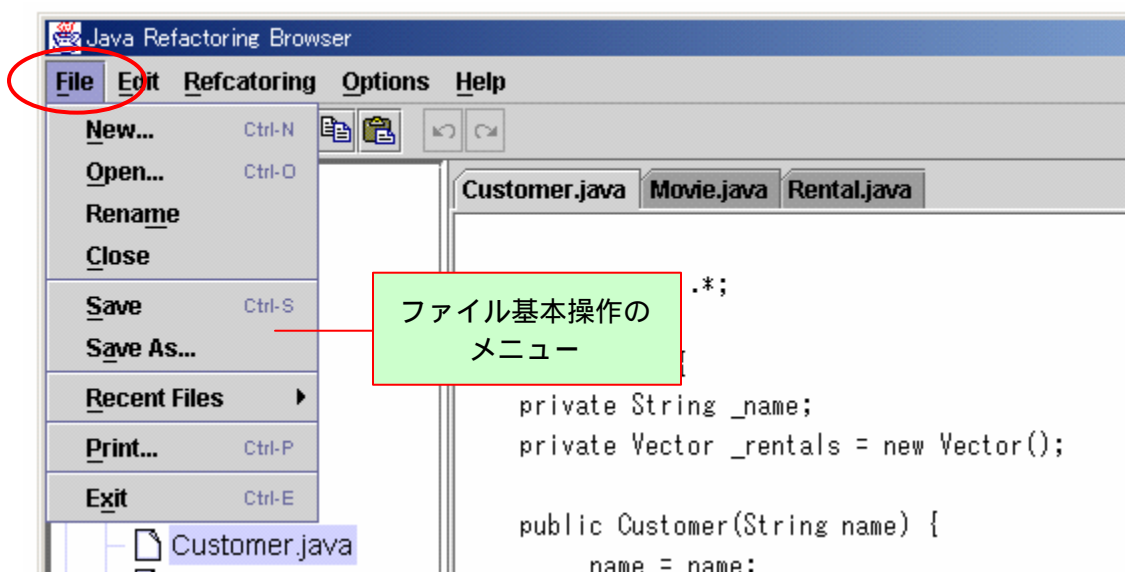
最前面のソースコード内に存在するクラスとメソッドを表示する。表示形式は、「クラス名#メソッド名(引数の型)」である。

3.2 メニュー

JRB は、File, Edit, Refactoring, Options, Help の 5 つのメニューを持つ。それぞれのメニューに対する操作方法を 3.2.1 ~ 3.2.5 に示す。

3.2.1 File メニュー (ファイルの基本操作)

ファイルに関する基本操作として、次の図に示す 9 つの機能を提供する。



(1) 新規作成(New)

ソースコード編集画面において、新規に Java ソースコードを作成する。

(2) オープン(Open)

既存の Java ソースコードをソースコード編集画面に読み込む。

(3) 名前の変更(Rename)

最前面のソースコードのファイル名を変更する。

(4) クローズ(Close)

最前面のソースコードを閉じる。ソースコードに変更が存在する場合、変更を保存あるいは破棄の確認が行われる。

(5) 保存(Save)

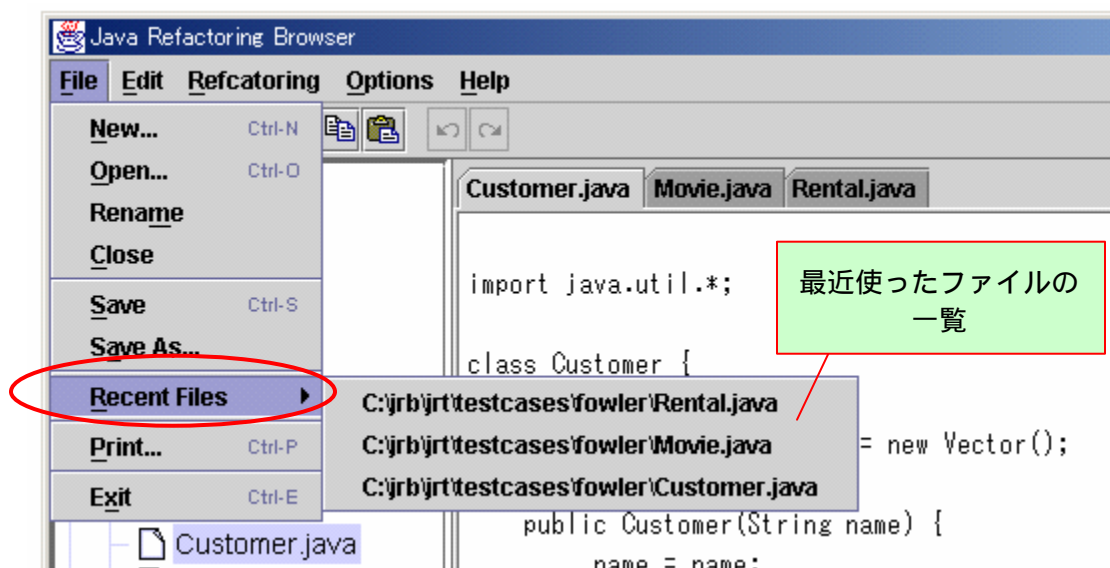
最前面のソースコードをファイルに上書き保存する。

(6) 名前を付けて保存(SaveAs)

最前面のソースコードを別のファイル名を付けて保存する。もとのソースコードは、ソースコード編集画面から削除され、新しいファイル名のソースコードが最前面に呼び出される。

(7) 最近使ったファイル(Recent Files)

ソースコードのオープンおよび名前を付けて保存を行った場合、そのソースコードのファイル名が Recent Files メニューのサブメニューに追加される。サブメニューにおいて、ファイルを選択することで、そのソースコードがオープンできる。



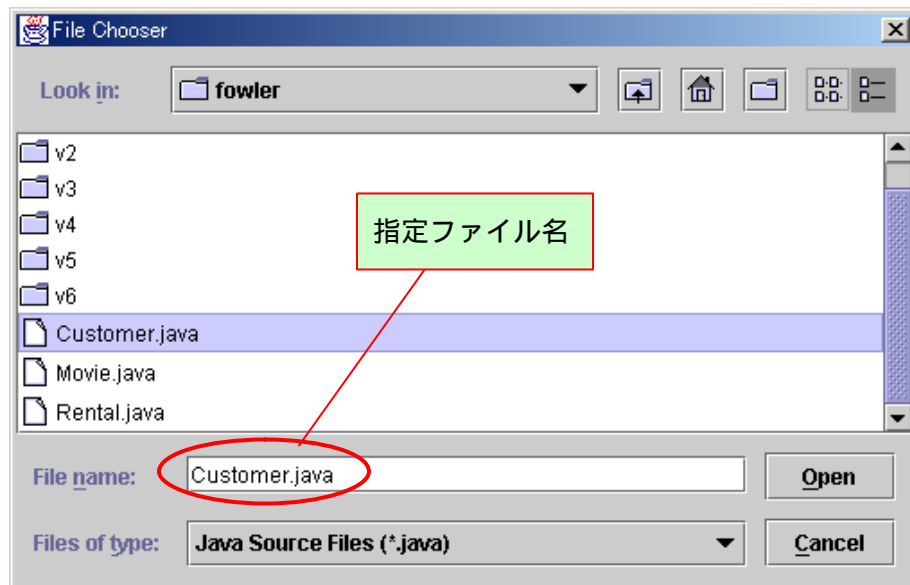
(8) 印刷(Print)

最前面のソースコードをプリンタに印刷する。

(9) 終了(Exit)

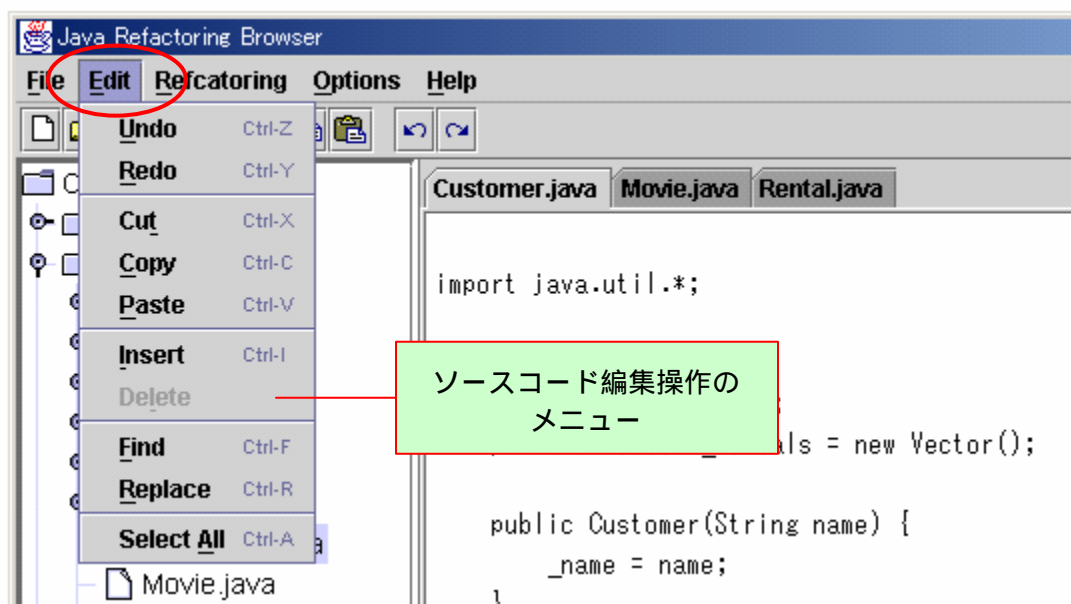
JRBを終了する。変更を含むソースコードに対しては保存するかどうか確認が行われる。

(1) 新規作成、(2) オープン (3) 名前の変更 (6) 名前を付けて保存において、ファイル名の指定は、次を示すファイル選択画面によって行う。



3.2.2 Edit メニュー（ソースコードの編集機能）

ソースコード編集画面上のソースコードに関する編集操作として、次に示す 10 つの機能を提供する。各操作は、最前面のソースコードに対して実行される。



（１）取り消し(Undo)

直前の編集操作を元に戻す。編集操作はソースコードごとに蓄積するため、各ソースコードは独立して元に戻すことができる。また、ソースコードに対する直接編集（追加、削除）だけでなく、リファクタリング操作を戻すこともできる。ただし、リファクタリング操作により変換された複数のソースコードに対して同時に取り消しを適用したい場合は、後述するリファクタリングメニューの取り消しを行うほうが良い。

(2) やり直し(Redo)

直前の取り消しをやり直す。取り消しと同様に、各ソースコードに対して独立に実行することができる。また、ソースコードに対する直接編集（追加、削除）だけでなく、リファクタリング操作の取り消しをやり直すこともできる。

(3) 切り取り(Cut)

選択テキストを削除し、そのテキストを（OS の提供する）クリップボードに格納する。

(4) コピー(Copy)

選択テキストをクリップボードにコピーする。テキストの削除は行わない。

(5) 貼り付け(Paste)

クリップボードに格納されているテキストを、ファイル上のカーソル位置に挿入する。

(6) テキストの挿入(Insert)

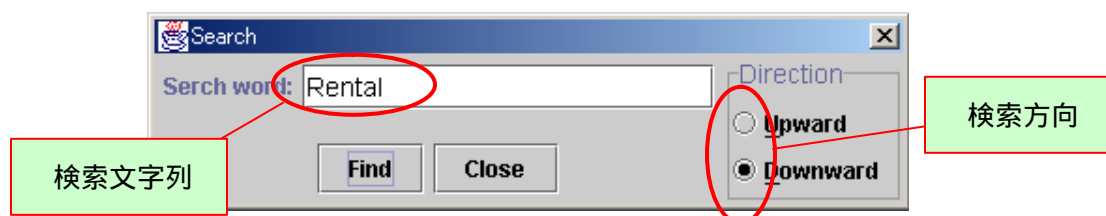
別ファイル内のテキストを、ソースコード上のカーソル位置に挿入する。挿入テキストの指定方法は、3.2.1 のファイルのオープンと同様である。

(7) 削除>Delete)

選択テキストを削除する。クリップボードに格納されているテキストは更新されない。

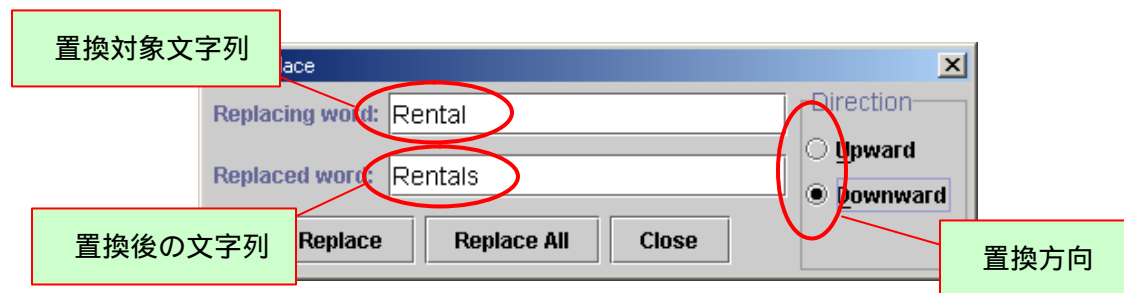
(8) 検索(Find)

ソースコードにおいて、特定の文字列を検索し、一致する文字列をハイライトで表示する。検索メニューを選択すると、以下のダイアログが現れるので、Search word に検索文字列を入力して Find ボタンを押す。検索方向は右側の Direction チェックボックスで指定する（ソースコードにおいて画面の上側の検索には Upward、下側には Downward を選択する）。



(9) 置換(Replace)

ソースコードにおいて、特定の文字列を新しい文字列に置換する。置換メニューを選択すると、以下の画面が現れるので、Replacing word に置換対象文字列を、Replaced word に置換後の文字列を入力して、Replace あるいは Replace All ボタンを押す。Replace All は一括置換である。置換方向についても、検索と同様に、Direction チェックボックスで指定する。

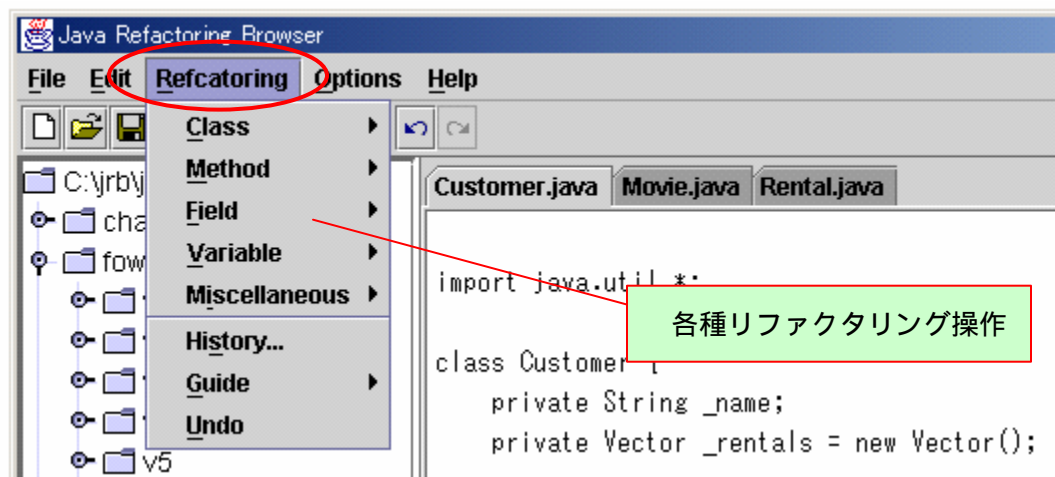


(1 0) すべて選択(Select All)

ファイル内部のテキストをすべて選択する .

3.2.3 Refactoring メニュー (リファクタリングの実行)

Java ソースコードに対するリファクタリング操作として、次に示す 8 つのメニュー項目 (5 種類のリファクタリング操作と 3 つの付随操作) を提供する . 各操作は、最前面のファイルに対して実行される .



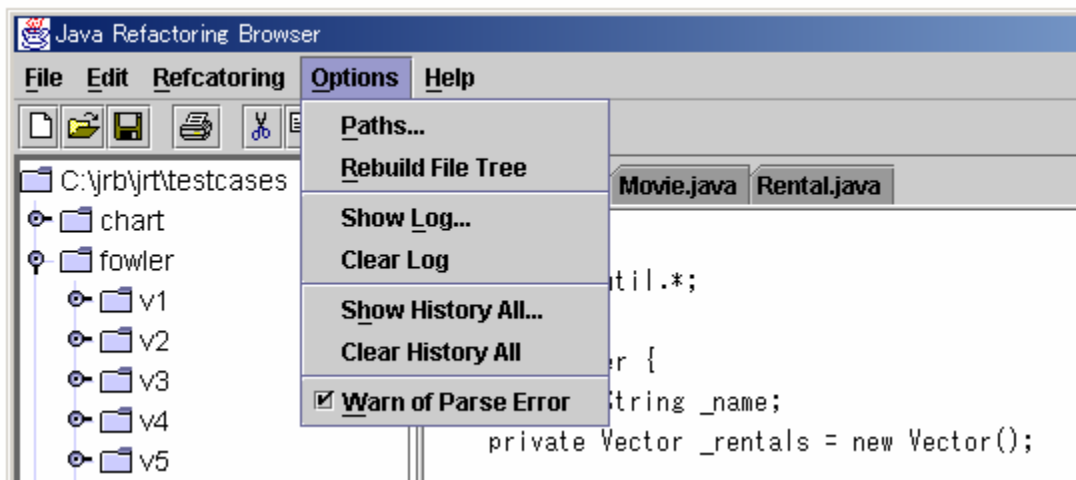
- (1) クラスに関するリファクタリング(Class)
- (2) メソッドに関するリファクタリング(Method)
- (3) フィールドに関するリファクタリング(Field)
- (4) ローカル変数に関するリファクタリング(Variable)
- (5) 上記に分類できないリファクタリング(Miscellaneous)
- (6) リファクタリング履歴の表示(History)
- (7) リファクタリング履歴の検索および次の操作に関する提案 (Guide)
- (8) リファクタリング操作のやり直し(Undo)

リファクタリング操作は、リファクタリングの種類により 5 つに分類されており、各メニュー項目はそれぞれ具体的なリファクタリング操作に対応するサブメニューで構成されている . これらの操作は、対象ソースコードにおける選択テキストあるいはカーソル位置に応じて適用可能なリファクタリング操作のみが選択可能となる (濃く表示される) . 反対に、選択不可能なメニューは淡

く表示され、選択できないようになっている。リファクタリング操作(1)～(5)およびそれに付随する操作(6)、(7)、(8)については、4章において詳しく述べる。

3.2.4 Options メニュー (オプション設定)

オプション設定に関して、次に示す7つの機能を提供する。



(1) パス設定(Paths)

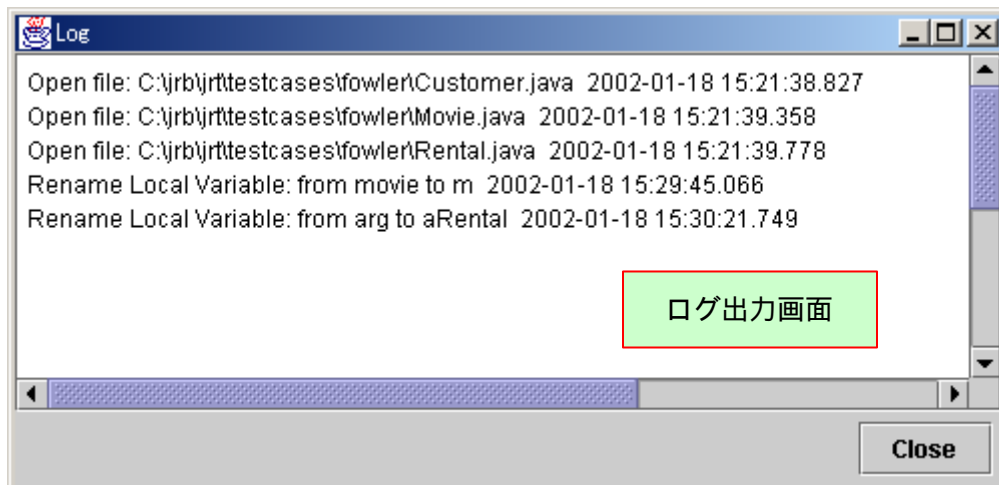
2.1で説明した環境変数を再設定する。Root directory を変更した場合、ファイル選択画面上の一覧が再構成される。

(2) ファイル選択画面上の一覧の再構成(Rebuild File Tree)

OS上のファイル構成との整合性をとるために、現在の一覧を強制的に破棄し、ファイル構成を検査して一覧を表示する。たとえば、Root directory 階層下に、別のディレクトリ階層下からファイルを移動・コピーした際に実行する。

(3) ログ画面の表示(Show Log)

次ページに示すログ出力画面を表示する。ログの形式は、「操作」+「実行時刻」である。ログの記録は、JRBの起動と同時に開始され、JRBを終了すると消去される。



(4) ログの消去(Clear Log)

ログを強制的に消去する。

(5) リファクタリング履歴の表示(Show History All)

過去に蓄積したすべてのリファクタリング操作を表示する（未保存のソースコードに対する履歴は表示しない）。履歴はJRBを終了した際に、ファイルとして記録され、起動時に再度読み込まれる。よって、履歴ファイルを直接消去するか、履歴消去操作を行わない限り過去の全操作（ファイルに保存されている履歴と保存済みソースコードに関する操作）が表示される。リファクタリング履歴の表示形式は、4.6においてリファクタリング操作の説明とともに述べる。

(6) リファクタリング履歴の消去(Clear History All)

リファクタリング履歴をすべて消去する。

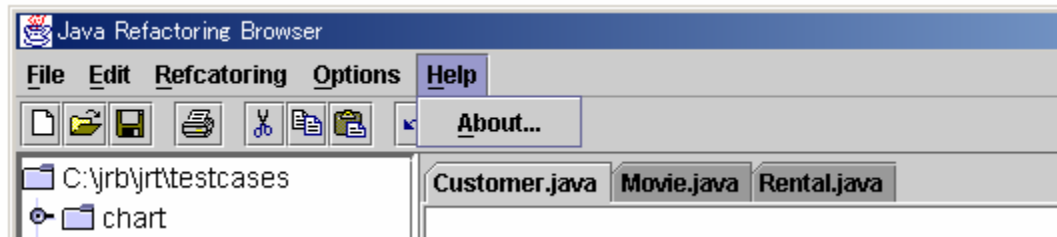
(7) 解析エラーの表示(Warn of Parse Error)

ソースコードの解析エラーをダイアログにより表示するかどうかを選択する。このメニュー項目にチェックを入れると、リファクタリング操作実行前（リファクタリングメニューの表示時）に解析エラーが発生した場合、その内容が表示される。チェックをはずすと、解析エラーは表示されない。表示される解析エラーの例を以下に示す。



3.2.5 Helpメニュー（ヘルプ）

ヘルプとして JRB に関する情報をダイアログにより表示する。



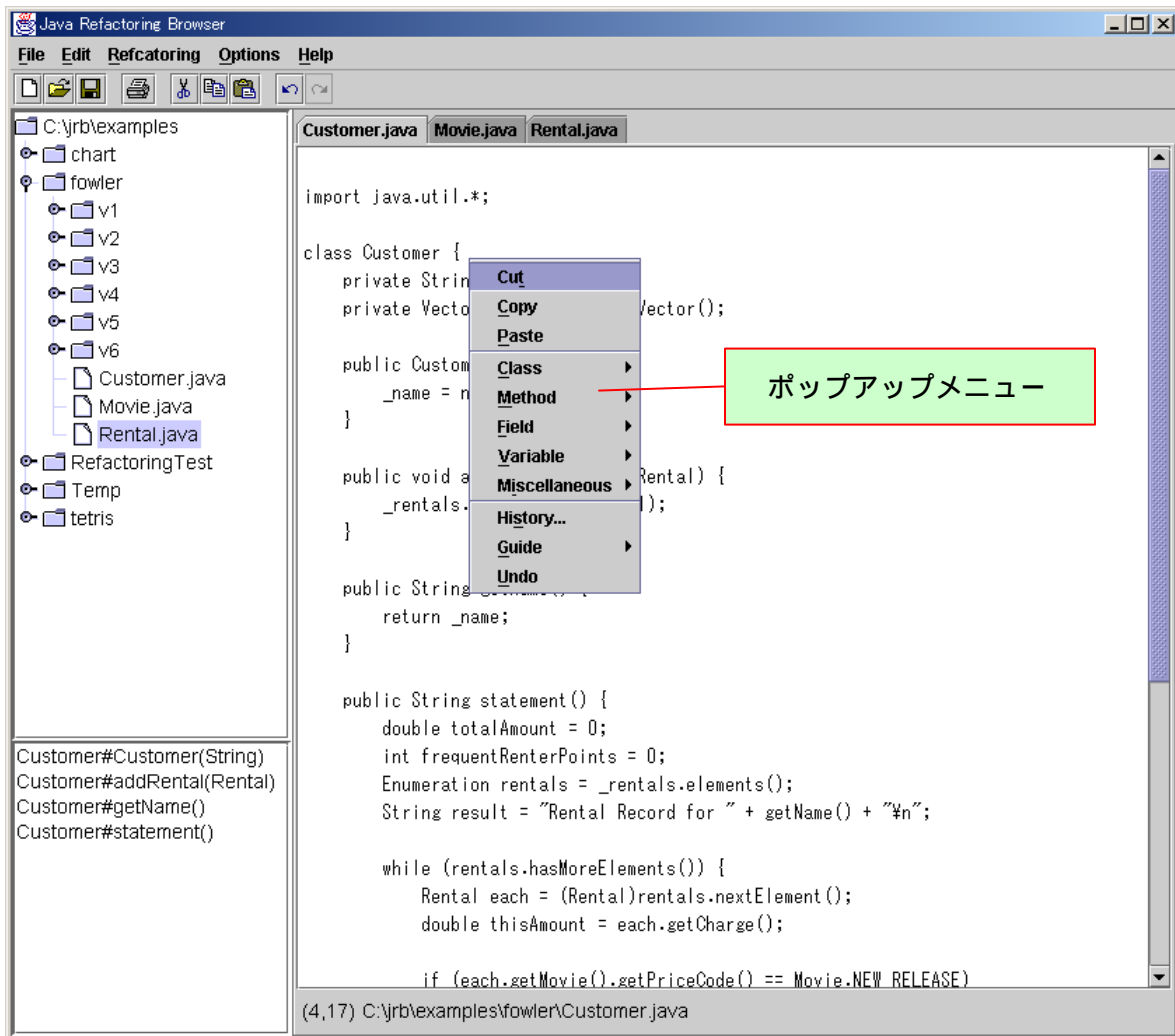
（１）JRB について(About)

バージョン番号，著作権，作者，連絡先などの情報を表示する。

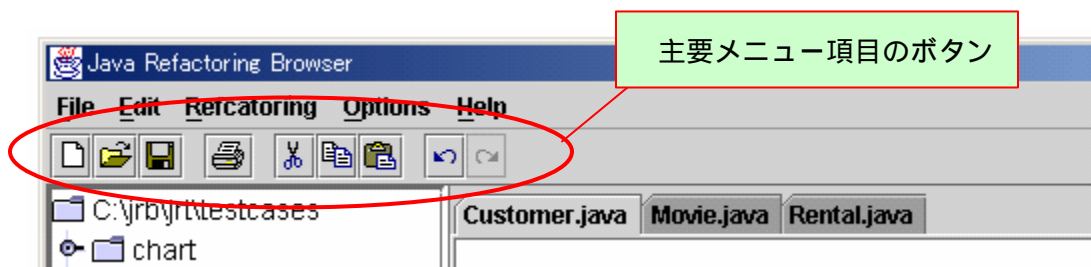


3.3 補足

JRB では操作性の向上のために、一部の編集メニューとリファクタリングメニューが、ソースコード編集画面において、マウスの右ボタンをクリックすることによってポップアップする。ポップアップ時の様子を以下に示す。



また、JRB では、主ないくつかの項目に関して、メニューの直下にボタンを用意している。ボタンを押した際の動作は、メニューで選択した際と同じである。以下に各ボタンとメニュー項目の対応を示す。



ボタンは、左側から順に、

- (1) ファイルの新規作成(File ? New)
- (2) ファイルのオープン(File ? Open)
- (3) ファイルの保存(File ? Save)
- (4) ファイルの印刷(File ? Print)
- (5) テキストの切り取り(Edit ? Cut)
- (6) テキストのコピー(Edit ? Copy)
- (7) テキストの貼り付け(Edit ? Paste)
- (8) 取り消し(Edit ? Undo)
- (9) やり直し(Edit ? Redo)

である。()内はメニューによる操作を表す。

さらに、メニュー項目によっては、ショートカット (ニーモニック) および (あるいは) メニューアクセラレータを提供する。

ショートカットに対応しているメニューでは、メニュー項目の英字 1 文字の下に下線が引かれている。メニュー項目を表示させた状態で、指定されている 1 文字をキーボードから入力することで、メニュー項目を起動できる。たとえば、File メニューを表示させた状態で、「N」を入力すると「ファイルの新規作成」が起動できる。

メニューアクセラレータを提供するメニュー項目は、メニュー項目が表示されていなくとも、メニュー項目の右側に表示されているキーでメニュー項目を起動できる。たとえば、「Ctrl-N」を入力する (Ctrl キーを押しながら N を押す) ことで、「ファイルの新規作成」が起動できる。

4. リファクタリング操作

3.2.3 で述べたように、JRB では、5 つに分類されるリファクタリング操作と 3 つの付随操作を提供している。JRB を用いたリファクタリングの流れは、図 2 のようになる。

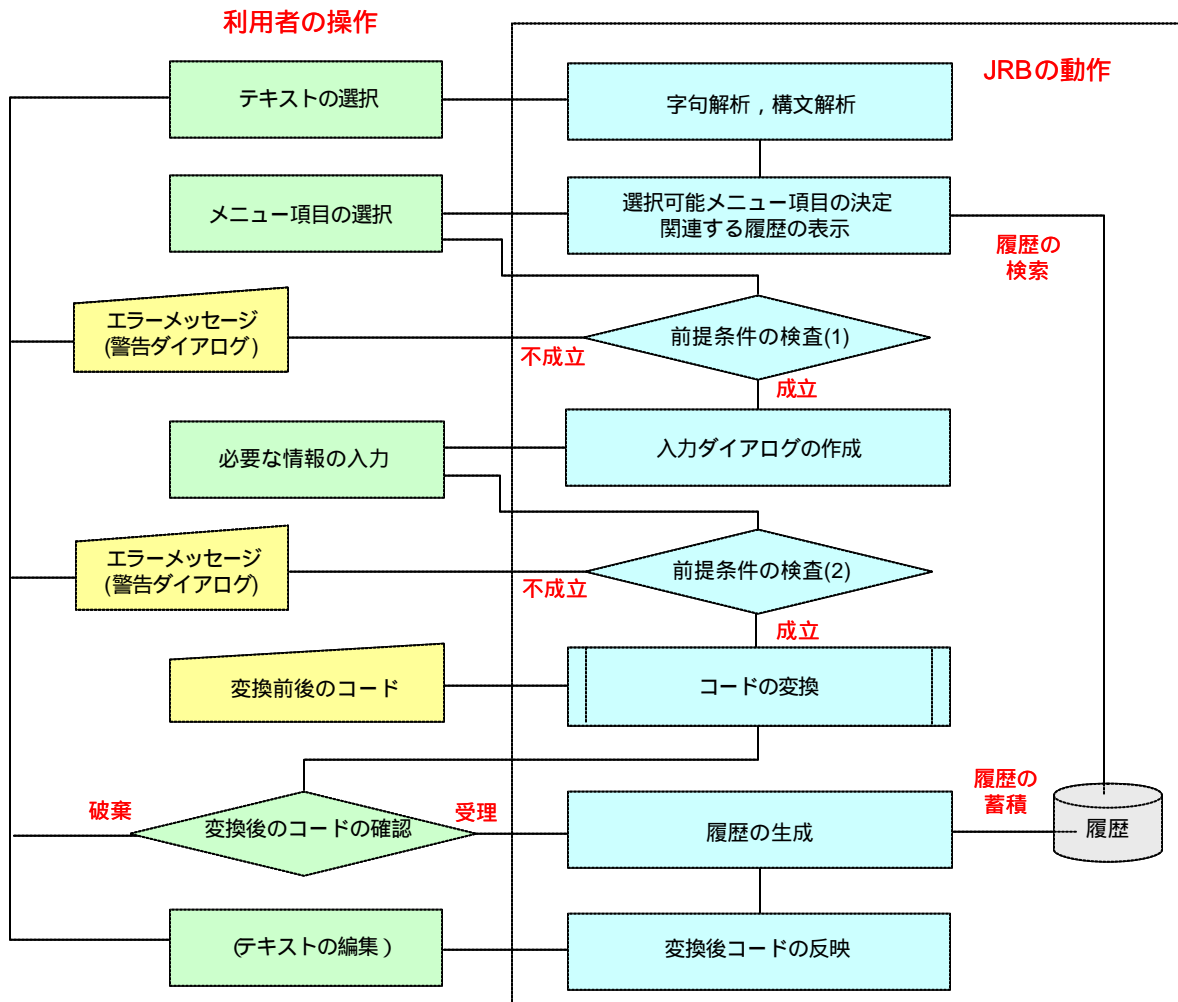


図2 JRBにおけるリファクタリングの流れ

利用者の行う手順をまとめると、次のようになる。

- (1) ソースコード編集画面において、テキストを選択する。
- (2) 表示されるメニュー項目から希望するリファクタリング操作を選択する。
- (3) リファクタリングが適用不可能な場合にエラーメッセージが出力されるので確認する。
- (4) 追加情報が必要な場合、対応する入力ダイアログが表示されるので、情報を入力する。
- (5) 入力情報に対して、リファクタリングが適用不可能な場合には、エラーメッセージが出力されるので確認する。
- (6) 変換前後のソースコードが別画面(変換確認画面)に表示されるので、変換後のソースコ

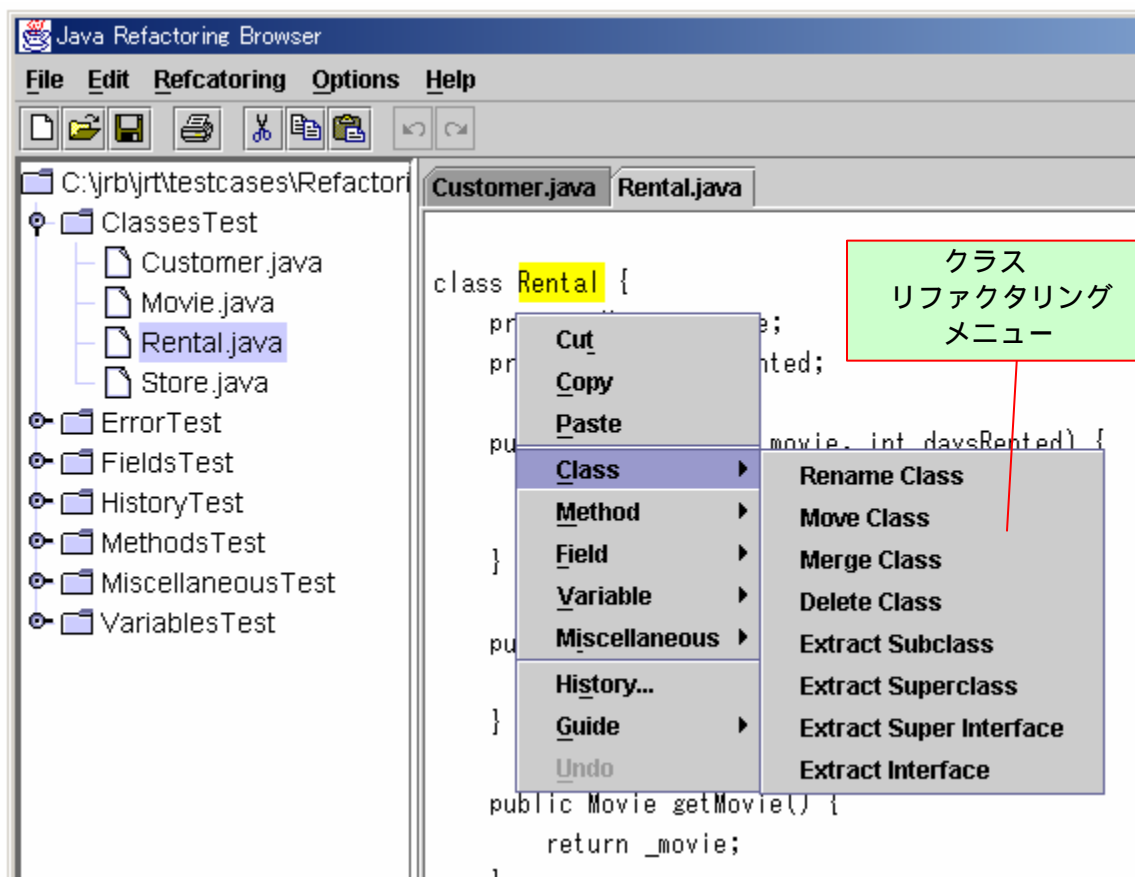
ードを受け入れるかどうかを決定する。

(7) 変換後のソースコードに変更が必要な場合、テキストを編集する。開発時には、この時点でコンパイルすることを薦める。

ここで、各リファクタリング操作が実行可能となる前提条件は、「論文 5.1」を参照のこと。また、前提条件を満たさない場合のエラーメッセージは、「ソフトウェア設計書 4.3」を参照のこと。本章では、それぞれのメニュー項目に関して、サブメニューと操作手順を説明する。

4.1 Class メニュー (クラスに関するリファクタリング)

クラスに関するサブメニューは、テキスト編集画面においてクラス宣言を選択した状態で表示され、実行可能となる。JRB では、次に示す 8 つのクラスリファクタリング操作を提供する。



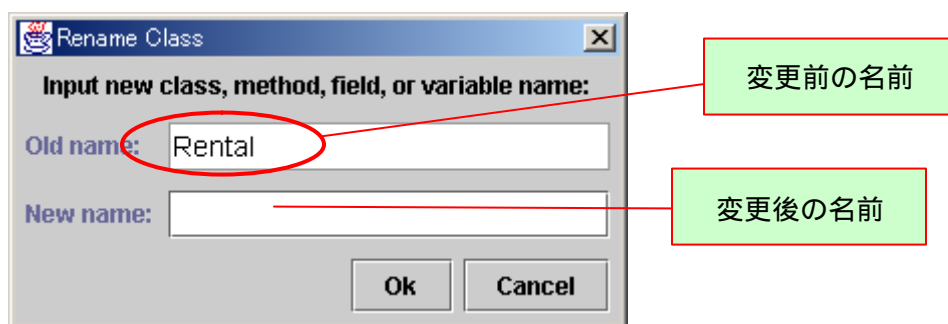
- (1) クラス名の変更(Rename Class)
- (2) クラスの移動(Move Class)
- (3) クラスの合併(Merge Class)
- (4) クラスの削除>Delete Class)
- (5) サブクラスの抽出(Extract Subclass)
- (6) スーパークラスの抽出(Extract Superclass)
- (7) スーパーインタフェースの抽出(Extract Super Interface)
- (8) インタフェースの抽出(Extract Interface)

それぞれのリファクタリング操作について、その実行手順を 4.1.1 ~ 4.1.8 に示す。

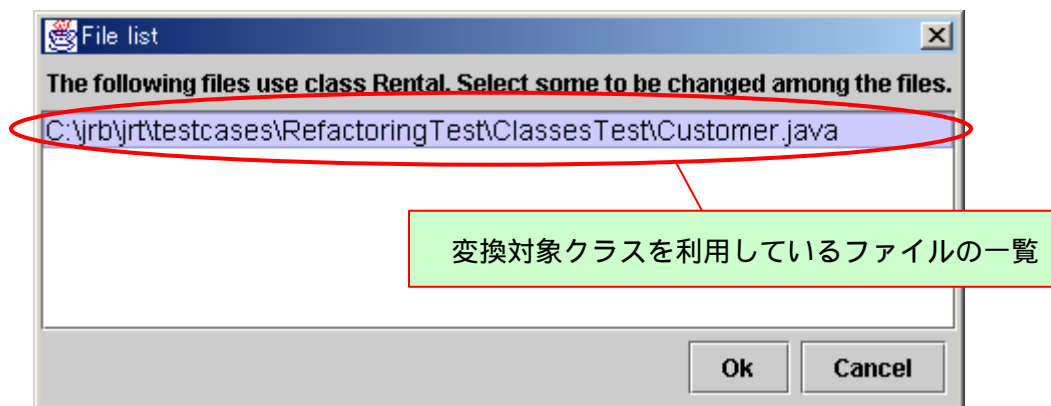
4.1.1 クラス名の変更(Rename Class)

指定されたクラスの名前を変更する。手順は以下の通りである。

- (i) 新規クラス名の入力ダイアログが表示されるので、New Name に新しいクラスの名前を入力し、Ok ボタンを押す。変更を中止したい場合は、Cancel ボタンを押す。Old Name には変更前の名前が自動的に表示される（この文字列は変更不可能）



- (ii) 名前を変更するクラス（上記の Rental）を利用している別のクラスを検索し、該当するクラスが探索ディレクトリ内に見つかった場合、そのファイル名が次のダイアログにより表示される。

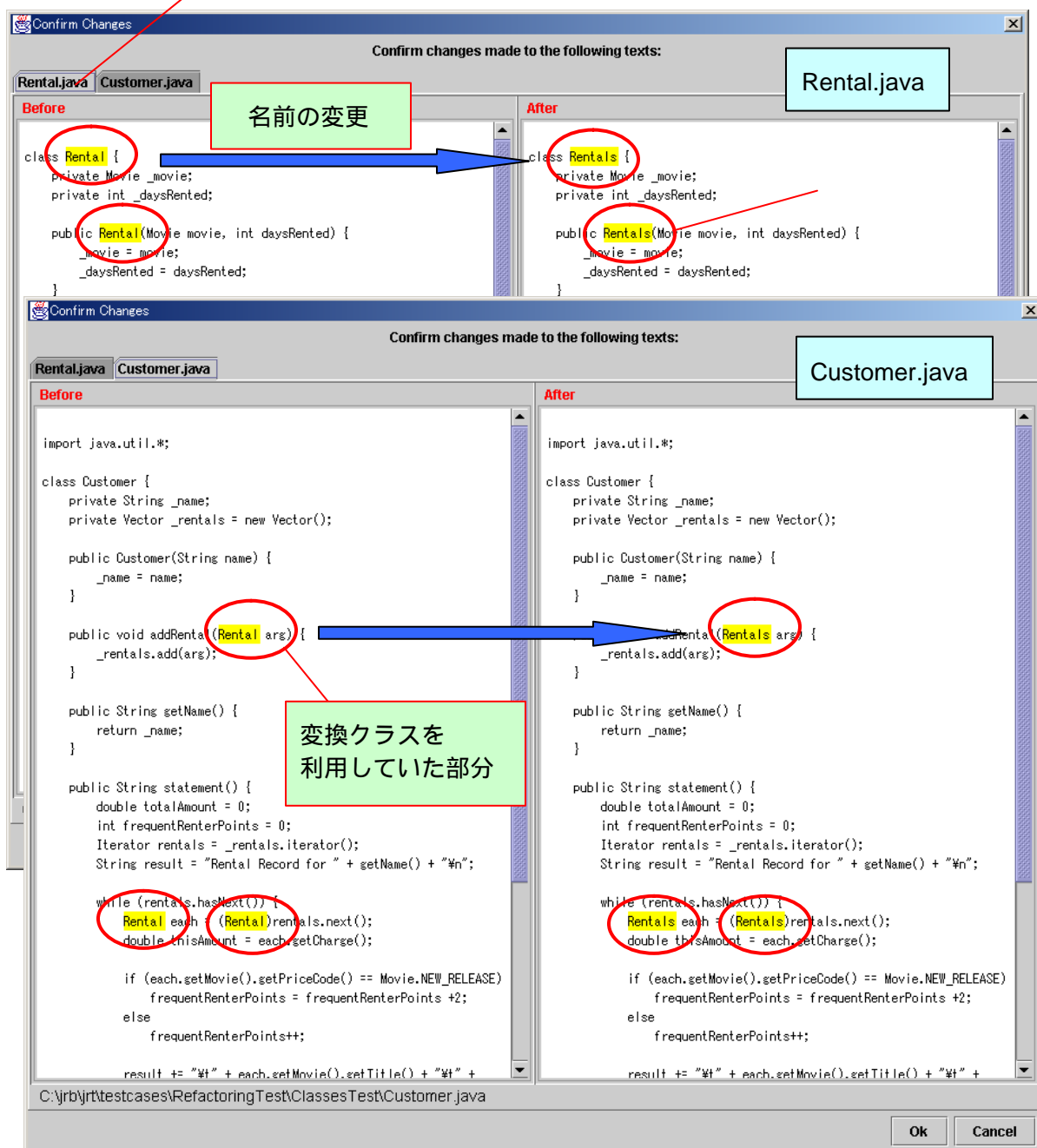


この一覧より、名前（参照名）を同時に変換するファイルを選択し、Ok ボタンを押す。上図は Customer.java ファイルを選択した例である。このダイアログでは、マウスのクリック（Ctrl+クリック）により複数ファイルの選択と選択解除が可能である。Cancel ボタンを押すと、既に選択されているファイルはクリアされ、ファイルを 1 つも選択せずに Ok ボタンを押した場合と同じ結果となる。

- (iii) 変換前後のソースコードがリファクタリング確認画面で表示される。変換前のソースコードが左側、変換後のソースコードが右側である。この画面において、変換が適用され

た部分は、ハイライト（実際のソフトウェアでは編集画面上で黄色）で表示される．利用者がこの変換を受理する場合は Ok ボタンを，破棄する場合は Cancel ボタンを押す．Ok ボタンを押すと，変換後のソースコードがソースコード編集画面に読み込まれる．

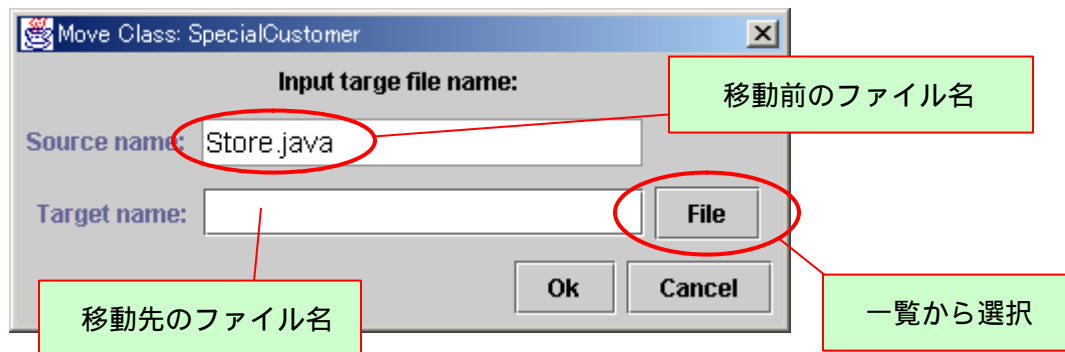
Rental.java ファイルの Rental クラスの名前を Rentals に変更した例を以下に示す．直接選択した Rental.java と同時変換を指定した Customer.java の変換前後のコードが確認画面に表示される（タブにより指定ファイルを前面に呼び出せる）．



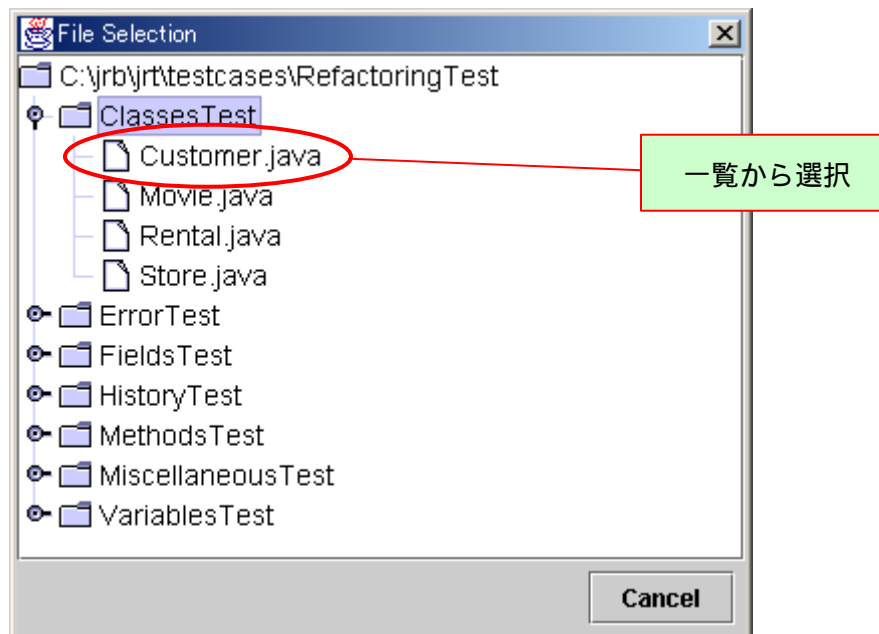
4.1.2 クラスの移動(Move Class)

指定されたクラスを異なるファイルに移動する。手順は以下の通りである。

- (i) 移動先指定ダイアログが表示されるので、Target name に移動先のファイル名を入力し、Ok ボタンを押す。入力したファイルが存在しない場合は、新規にファイルが生成される。移動を中止したい場合は、Cancel ボタンを押す。Source Name には変更前の名前が自動的に表示される（この文字列は変更不可能）。

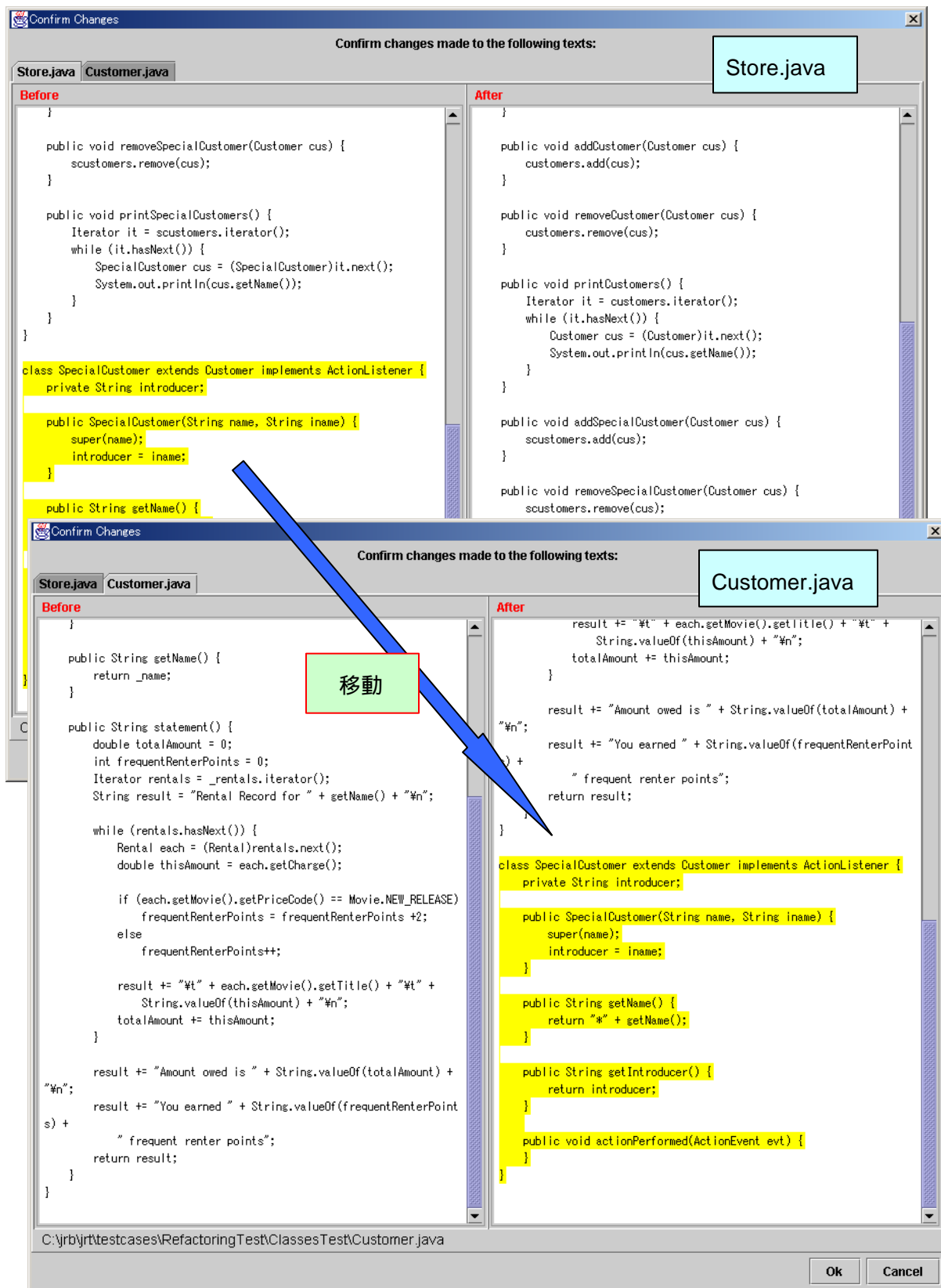


このダイアログにおいて、File ボタンを押すと、次に示すファイル選択ダイアログにより、探索ディレクトリ内に存在するファイルの一覧から移動先のファイルを選択することができる。



- (ii) クラスの移動にともない、移動クラスを利用できなくなった場合（たとえば、クラスを別パッケージに移動した場合）、該当クラスを含むファイルの一覧が、参考のために表示される（同時変換は行わない）。

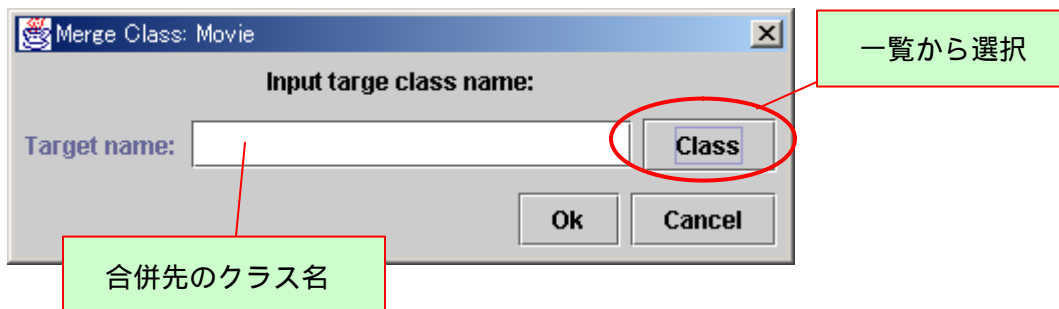
Store.java ファイル内の SpecialCustomer クラスを Customer.java ファイルに移動した際の実行結果を以下に示す。



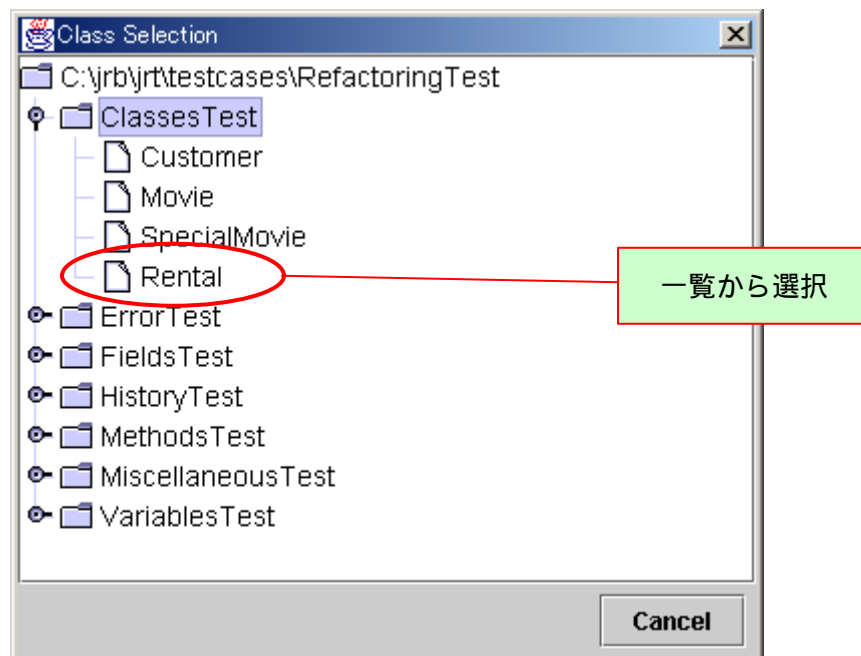
4.1.3 クラスの合併(Merge Class)

指定されたクラスの方法とフィールド全体を異なるクラスに挿入する。手順は以下の通りである。

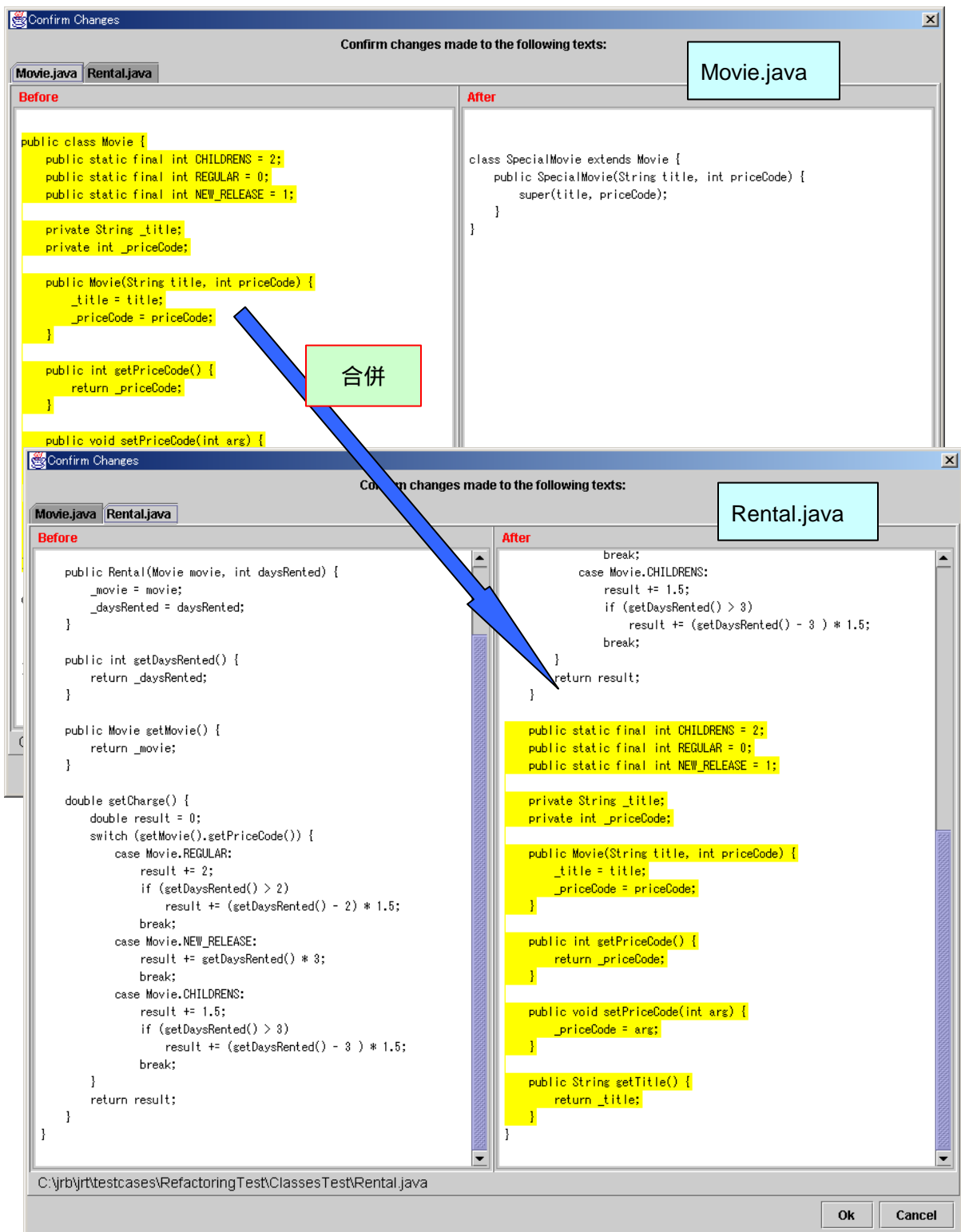
- (i) クラス名入力ダイアログが表示されるので、Target name に合併先のクラス名を入力し、Ok ボタンを押す。合併を中止したい場合は、Cancel ボタンを押す。クラス名の指定形式は、「クラス名」あるいは「ファイル名#クラス名」である。クラス名だけを指定した場合、合併元のクラスと同じパッケージに存在するクラスが合併先クラスとして探索される。



このダイアログにおいて、Class ボタンを押すと、次に示すようなクラス選択ダイアログにより、探索ディレクトリに存在するクラスの一覧から移動先のクラスを選択することができる。



Movie.java ファイルの Movie クラスを Rental.java ファイルの Rental クラスに合併した際の実行結果を以下に示す。

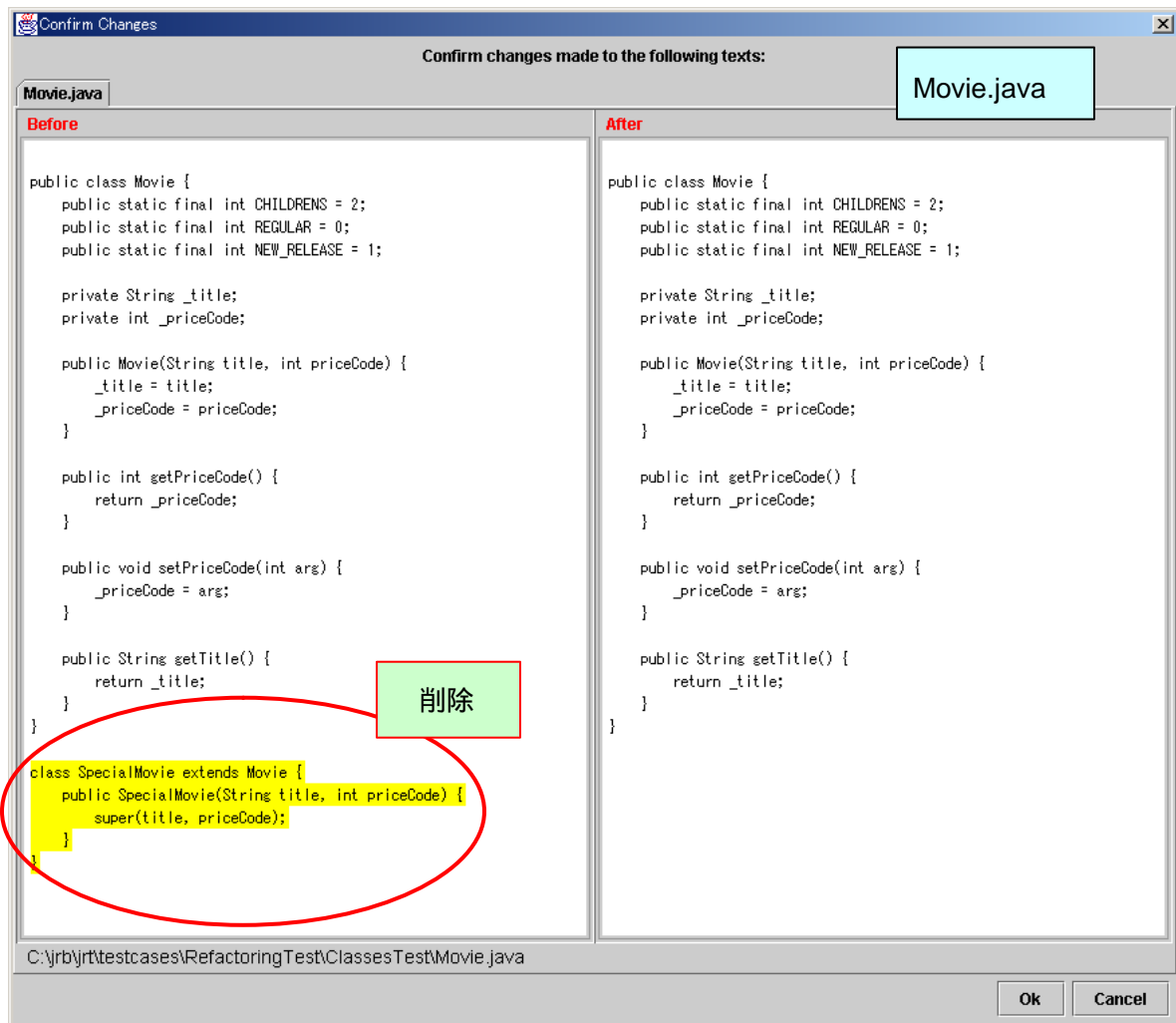


4.1.4 クラスの削除>Delete Class)

指定されたクラスを削除する。手順は以下の通りである。

- (i) 削除するクラスを利用しているクラスが探索ディレクトリ内に存在しない場合のみ削除が実行される。利用クラスが存在する場合は、警告ダイアログが表示され、削除は中止される。

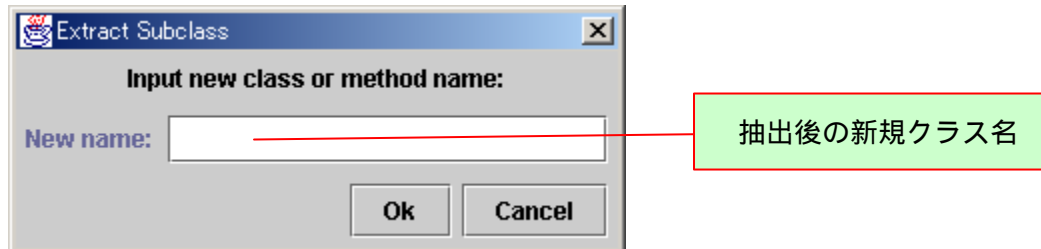
Movie.java ファイル内の SpecialMovie クラスを削除した際の実行結果を以下に示す。



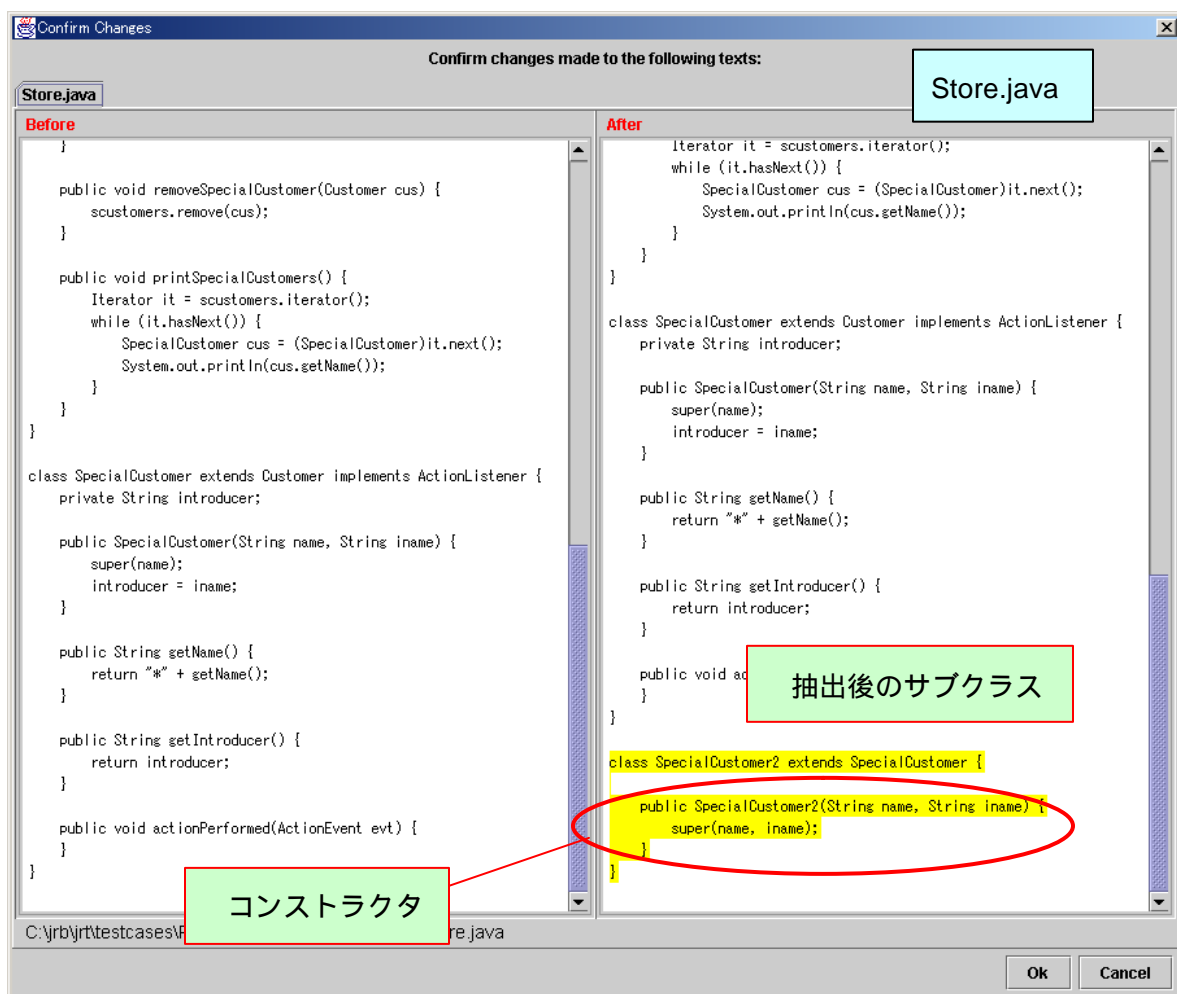
4.1.5 サブクラスの抽出(Extract Subclass)

指定されたクラスの子クラスの宣言を新規に作成する。手順は以下の通りである。

- (i) クラス名入力ダイアログが表示されるので、New name に抽出後の新規クラスの名前を入力し、Ok ボタンを押す。抽出を中止したい場合は、Cancel ボタンを押す。



サブクラスの抽出においては、もとのクラスに存在するコンストラクタを呼び出す新規コンストラクタも同時に生成する。Store.java ファイルの SpecialCustomer クラスからサブクラス SpecialCustomer2 を抽出した際の実行結果を以下に示す。

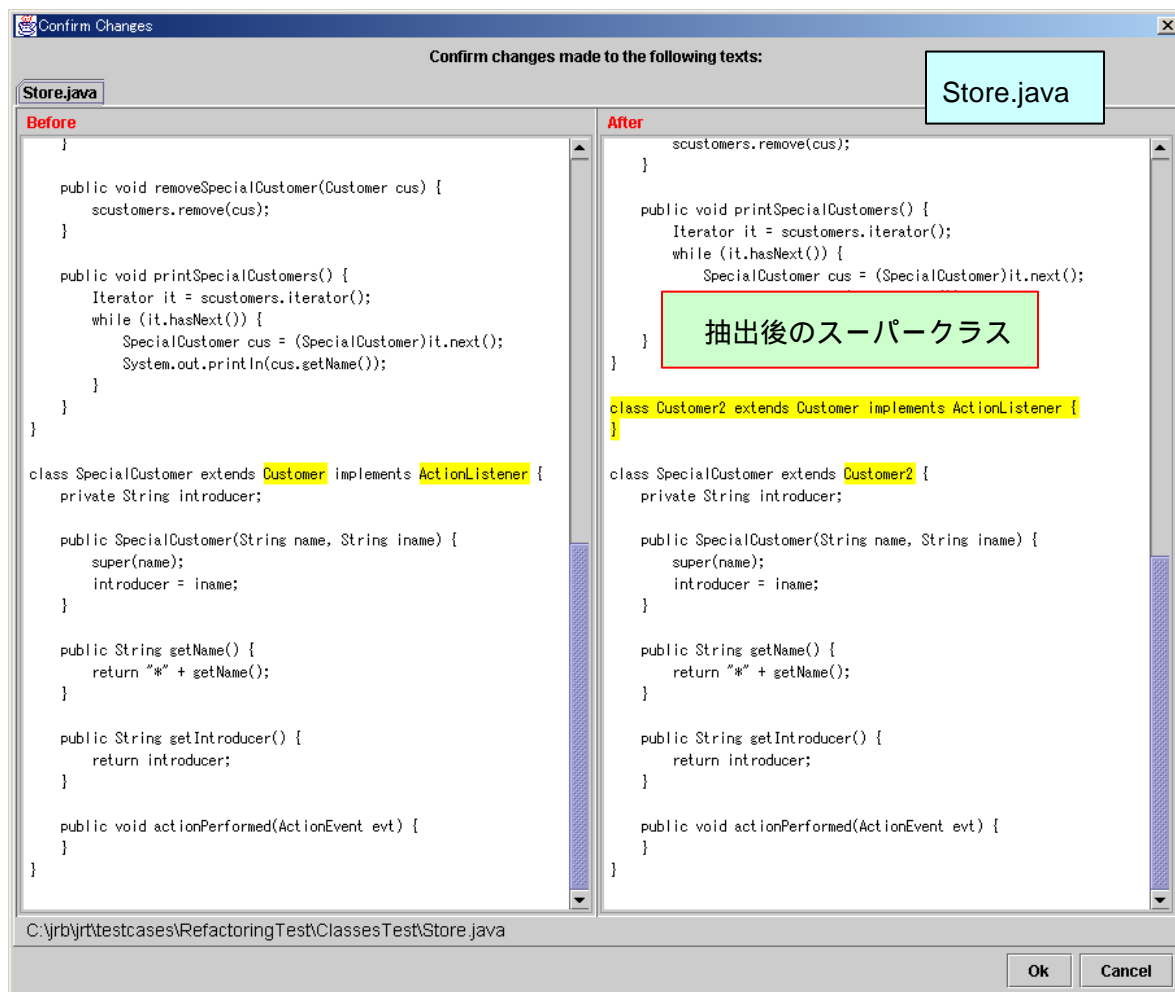


4.1.6 スーパークラスの抽出(Extract Superclass)

指定されたクラスのスーパークラスの宣言を新規に作成する。手順は以下の通りである。

- (i) 4.1.5 のサブクラスの抽出と同様の入力ダイアログが表示されるので、New name に抽出後のスーパークラスの名前を入力し、Ok ボタンを押す。もとのクラスは、新規に抽出したクラスのサブクラス(extends)となる。

Store.java ファイルの SpecialCustomer クラスからスーパークラス Customer2 を抽出した際の実行結果を以下に示す。

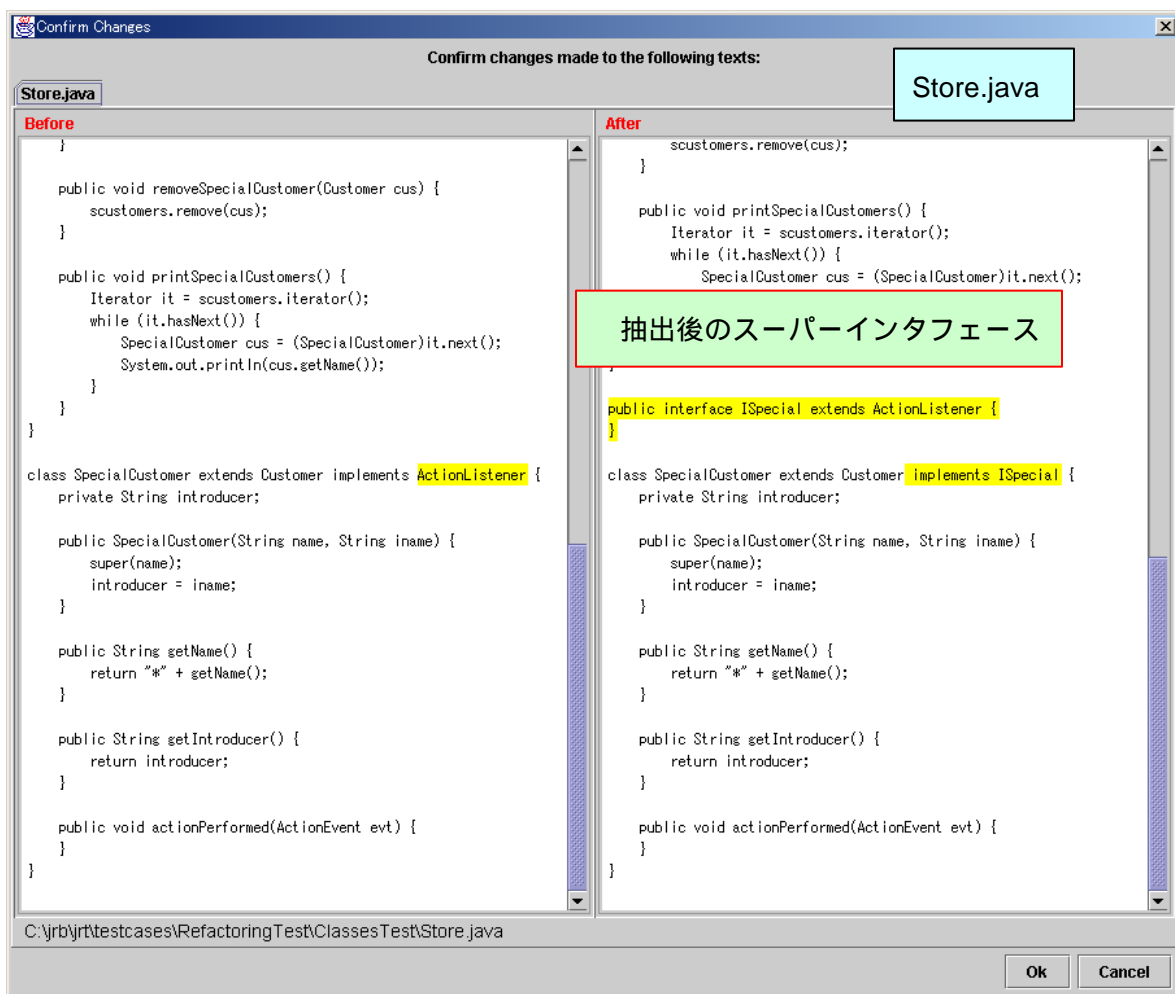


4.1.7 スーパーインタフェースの抽出(Extract Super Interface)

指定されたクラスのスーパーインタフェースの宣言を新規に作成する。手順は以下の通りである。

- (i) 4.1.5 のサブクラスの抽出と同様の入力ダイアログが表示されるので、New name に抽出後のスーパーインタフェースの名前を入力し、Ok ボタンを押す。もとクラスは、新規に抽出したインタフェースを implements 継承する。

Store.java ファイルの SpecialCustomer クラスからスーパーインタフェース ISpecial を抽出した際の実行結果を以下に示す。

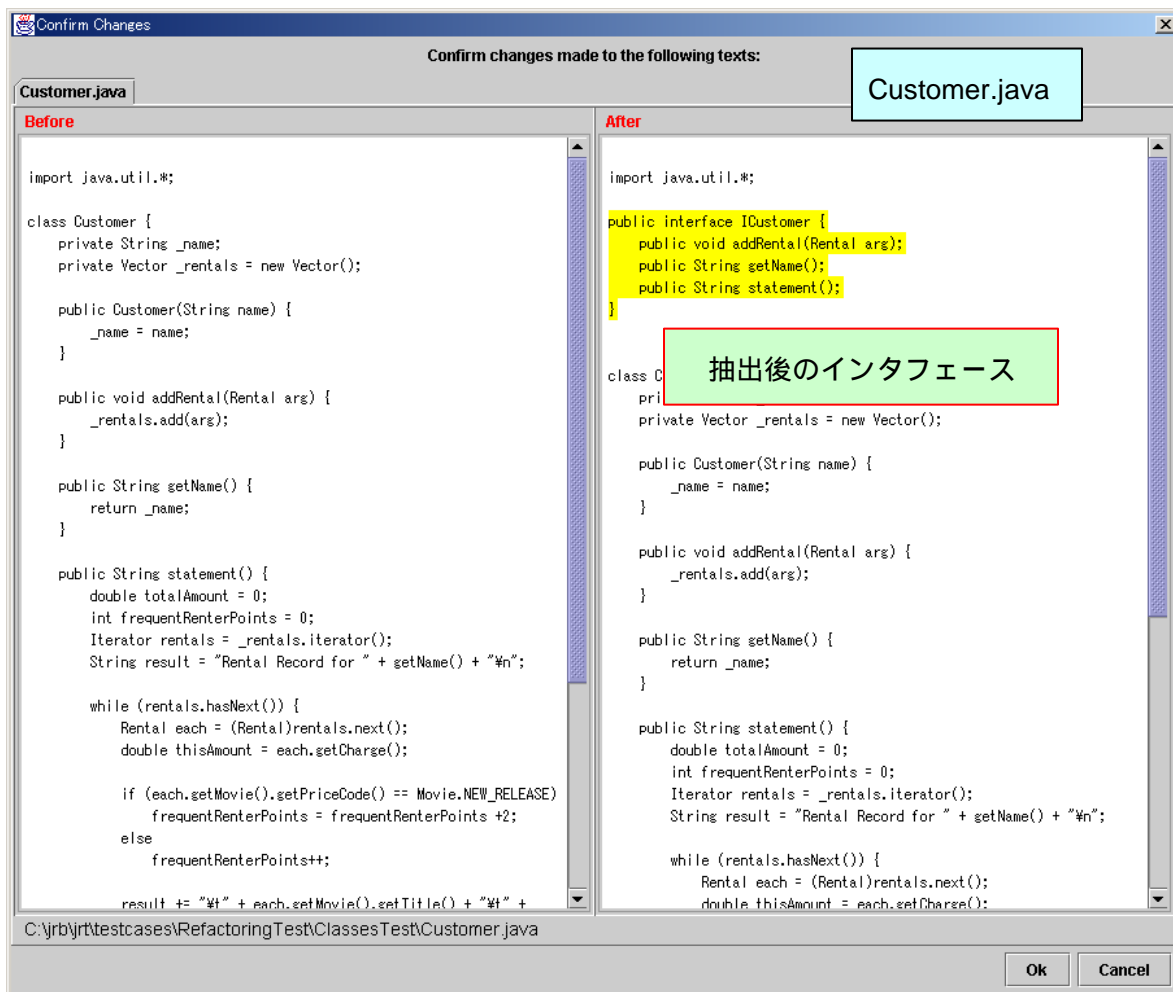


4.1.8 インタフェースの抽出(Extract Interface)

指定されたクラスのインタフェースを新規に作成する。手順は以下の通りである。

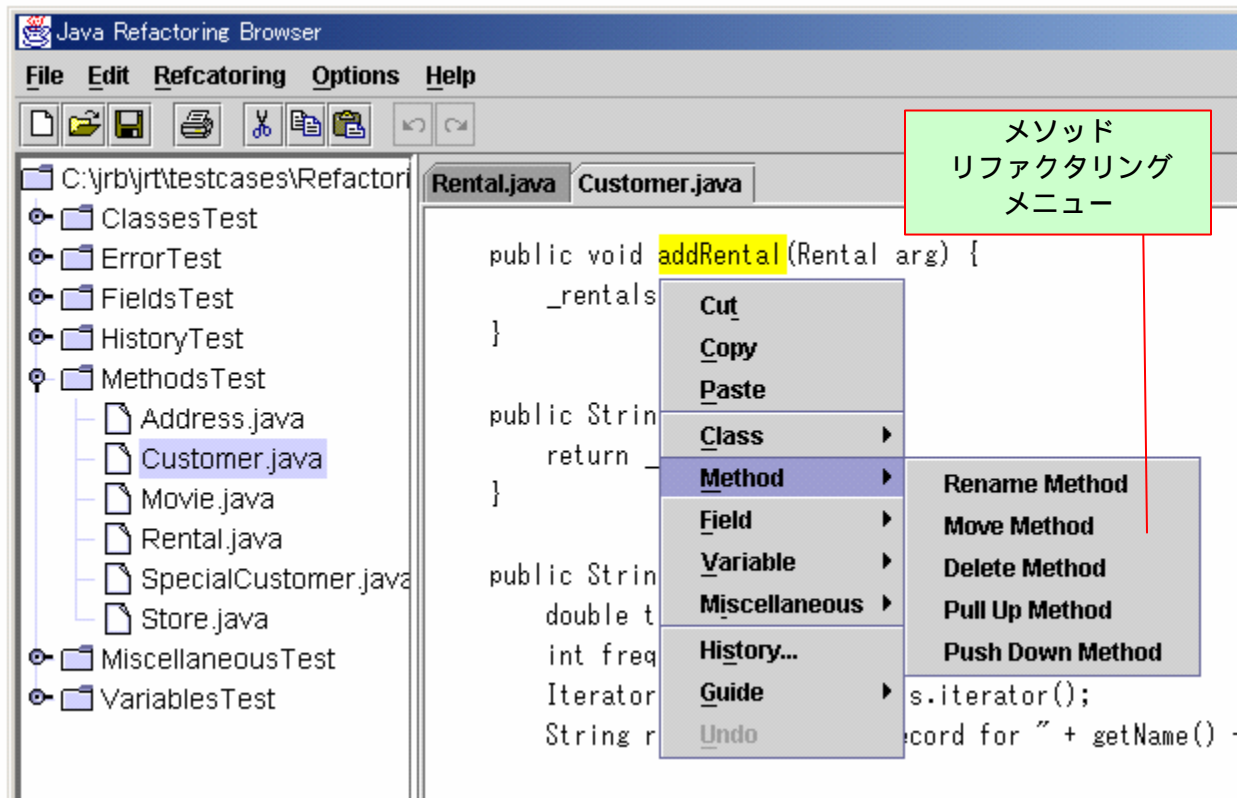
- (i) 4.1.5 のサブクラスの抽出と同様の入力ダイアログが表示されるので、New name に抽出後のインタフェースの名前を入力し、Ok ボタンを押す。

Customer.java ファイルの Customer クラスからインタフェース ICustomer を抽出した際の実行結果を以下に示す。



4.2 Methodメニュー（メソッドに関するリファクタリング）

メソッドに関するサブメニューは、ソースコード編集画面においてメソッド宣言を選択した状態で表示され、実行可能となる。JRBでは、次に示す5つのメソッドリファクタリング操作を提供する。



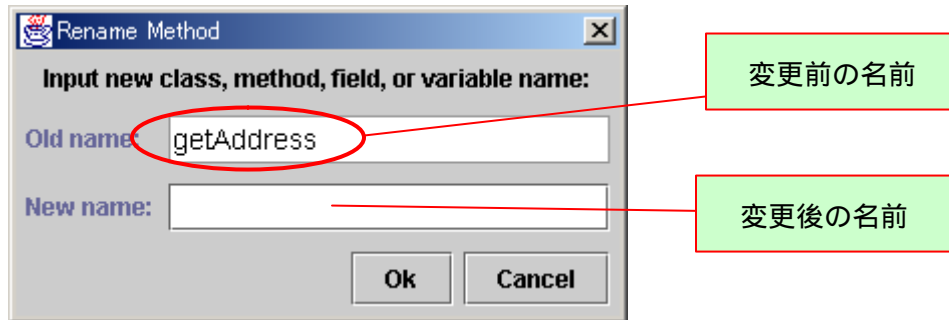
- (1) メソッド名の変更(Rename Method)
- (2) メソッドの移動(Move Method)
- (3) メソッドの削除>Delete Method)
- (4) メソッドの引き上げ(Pull Up Method)
- (5) メソッドの引き下げ(Push Down Method)

それぞれのリファクタリング操作について、その実行手順を4.2.1～4.2.5に示す。

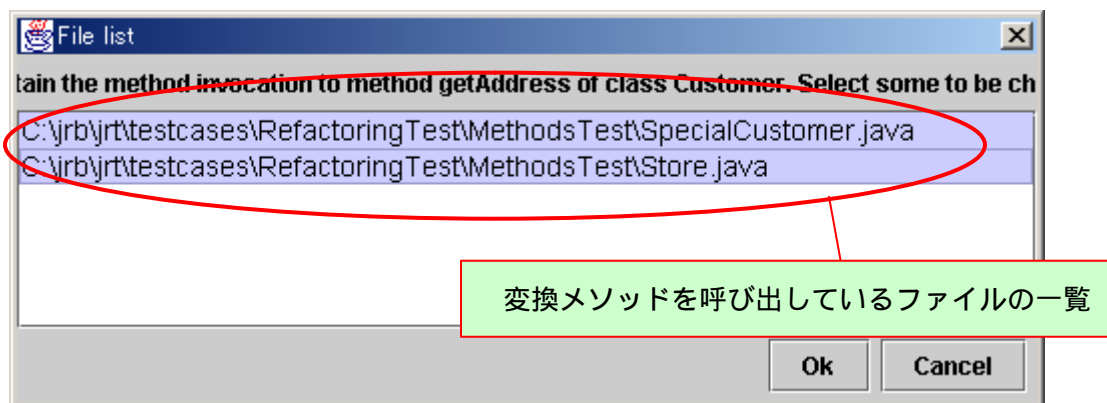
4.2.1 メソッド名の変更(Rename Method)

指定されたメソッドの名前を変更する。手順は以下の通りである。

- (i) メソッド名入力ダイアログが表示されるので、New Name に新しいメソッドの名前を入力し、Ok ボタンを押す。変更を中止したい場合は、Cancel ボタンを押す。Old Name には変更前の名前が自動的に表示される（この文字列は変更不可能）



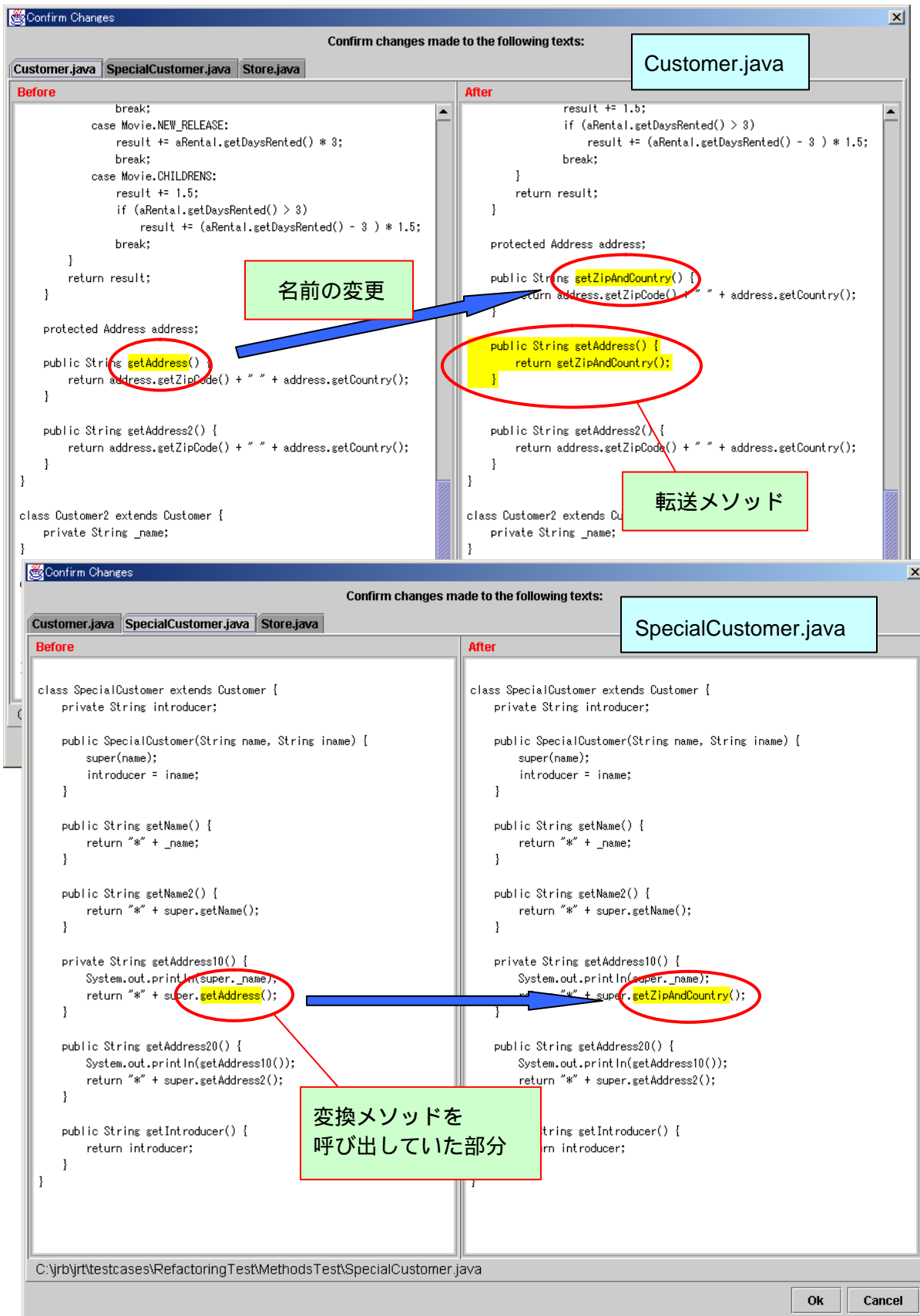
- (ii) 名前を変更するメソッドを呼び出しているクラスが探索ディレクトリ内に見つかった場合、次のようなファイル指定ダイアログが表示される。



この一覧より、呼び出しメソッド名（参照名）を同時に変換するファイルを選択し、Ok ボタンを押す。上図は Customer.java ファイルと Store.java ファイルを選択した例である。

- (iii) 変換前後のソースコードが確認画面に表示される。名前変更時には、変更前の名前でも呼び出し可能とするため、もとのメソッドにその処理を委譲あるいは転送するメソッドが生成される。

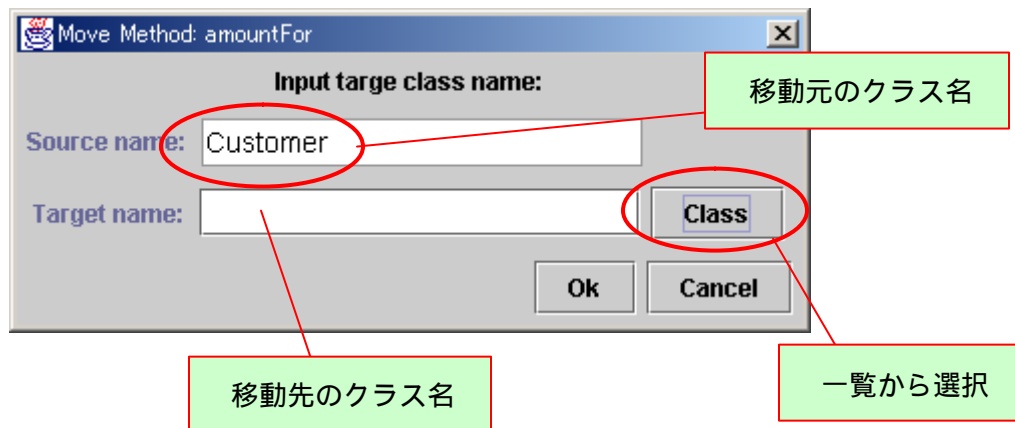
Customer.java ファイルの Customer クラスのメソッド getAddress を getZipAndCountry に変更した際の実行結果（Customer.java と SpecialCustomer.java のみ）を以下に示す。



4.2.2 メソッドの移動(Move Method)

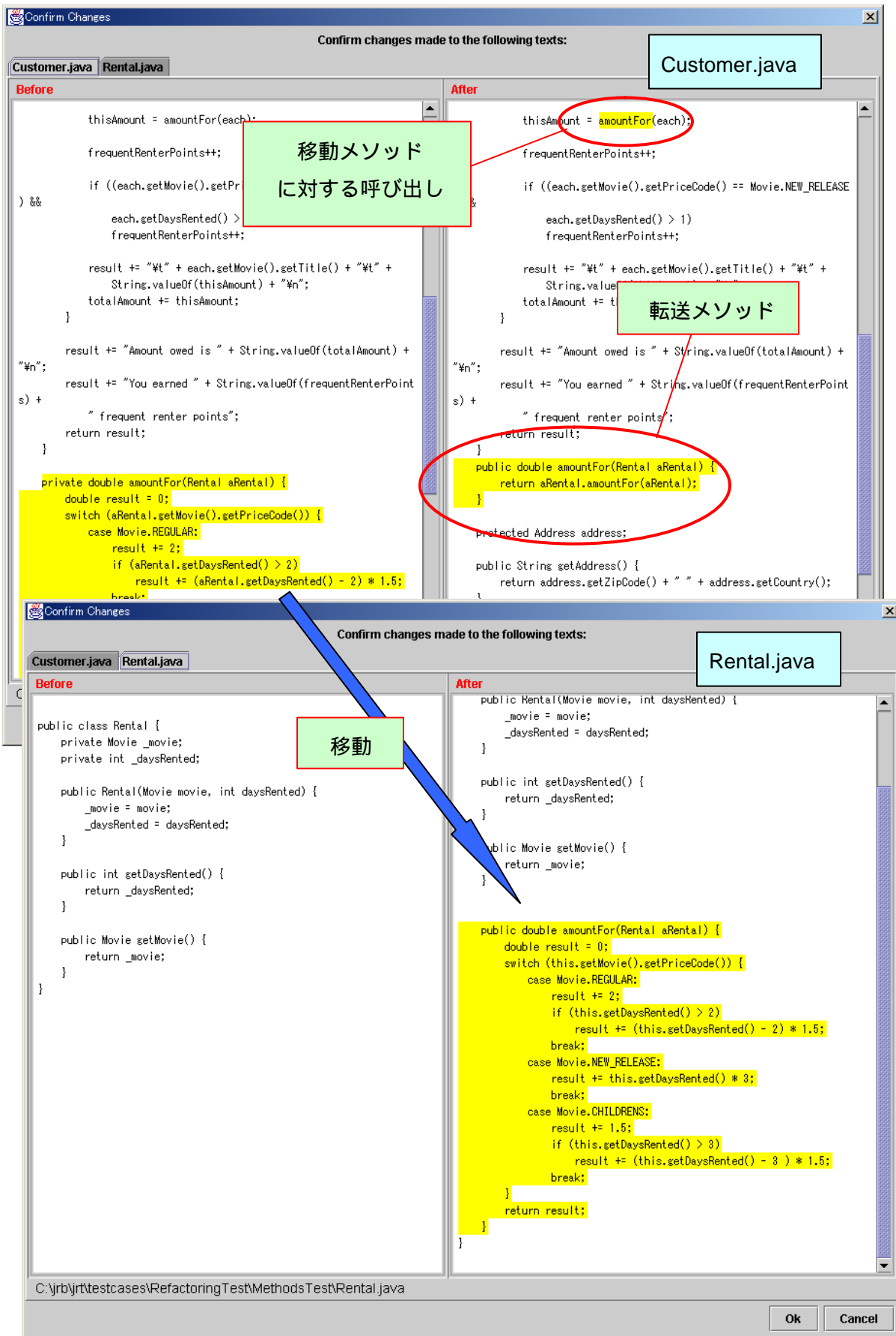
指定されたメソッドを異なるクラスに移動する。手順は以下の通りである。

- (i) 移動先クラス入力ダイアログが表示されるので、Target name に移動先のクラス名を入力し、Ok ボタンを押す。Source Name には移動前のクラス名が自動的に表示される（この文字列は変更不可能）。クラス名の指定形式は、4.1.3 と同様である。クラス名だけを指定した場合、移動メソッドを所有するクラスと同じパッケージに存在するクラスが移動先クラスとして探索される。



- (ii) 移動メソッド内で利用されるフィールドあるいはパラメータより、移動先クラスの参照オブジェクトの名前を決定する。参照名が決定できた場合、移動先のメソッドに処理を移譲するメソッド（移動先のメソッドを単に呼び出す委譲メソッドあるいは転送メソッド）が、移動元クラスに生成される。
- (iii) 移動メソッドを呼び出しているメソッドが探索ディレクトリ内に存在する場合、このメソッドを含むファイルも同時に確認画面に表示される。

Customer.java ファイルの Customer クラスのメソッド amountFor を Rental.java ファイルの Rental クラスに移動した際の実行結果を以下に示す。

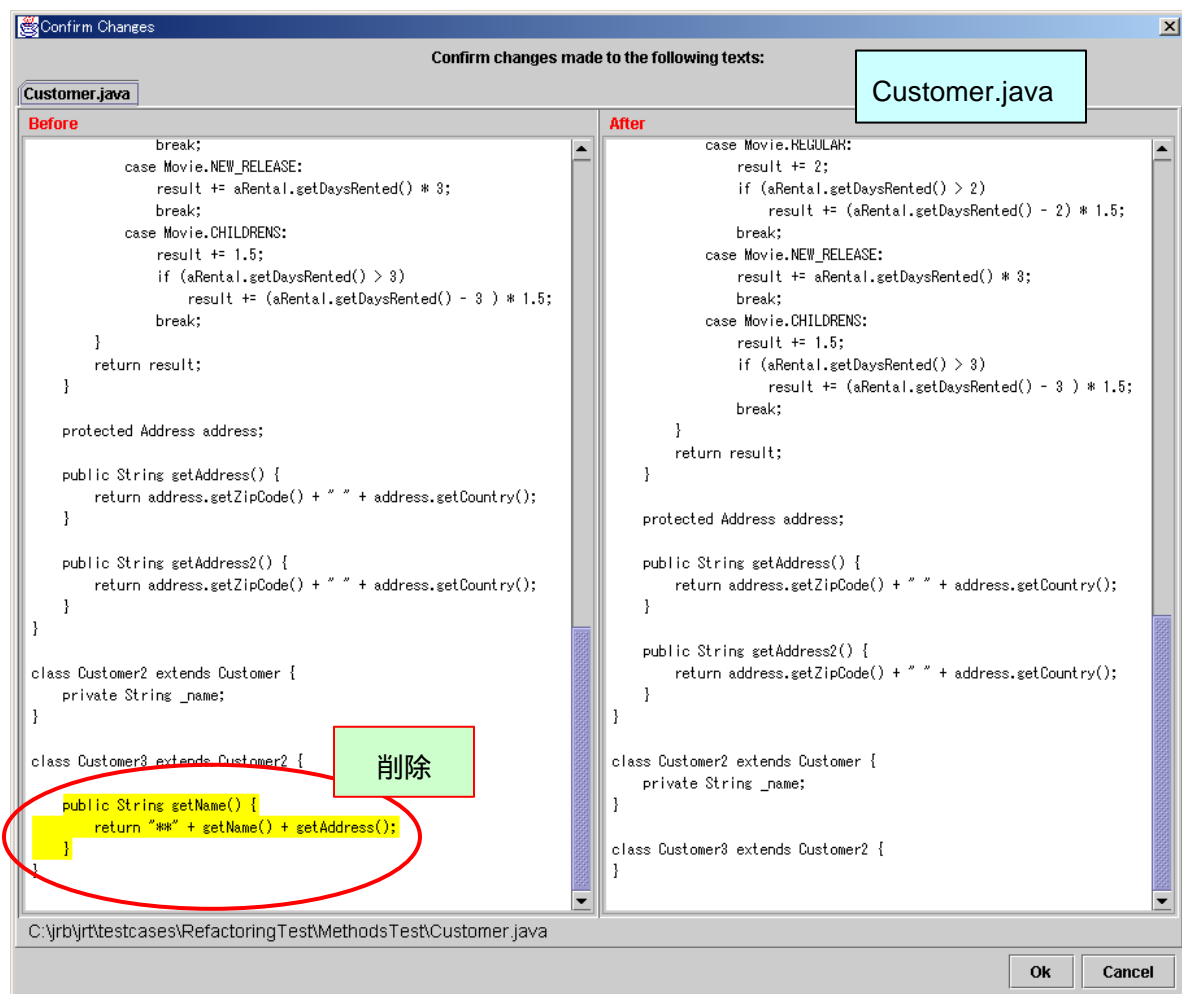


4.2.3 メソッドの削除>Delete Method)

指定されたメソッドを削除する．手順は以下の通りである．

- (i) 削除するメソッドを呼び出しているメソッドが探索ディレクトリに存在しない場合のみ削除が実行される．呼び出しメソッドが存在する場合は，警告ダイアログが表示され，削除は中止される．

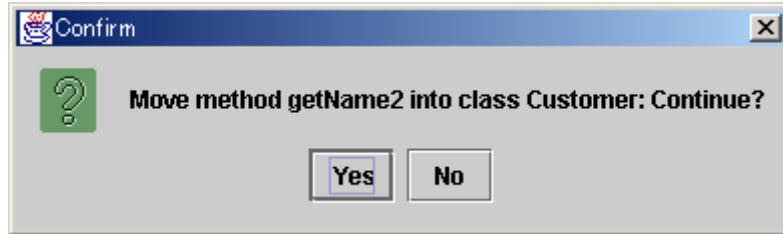
Customer.java ファイルの Customer3 クラスの getName メソッドを削除した際の実行結果を以下に示す．



4.2.4 メソッドの引き上げ(Pull Up Method)

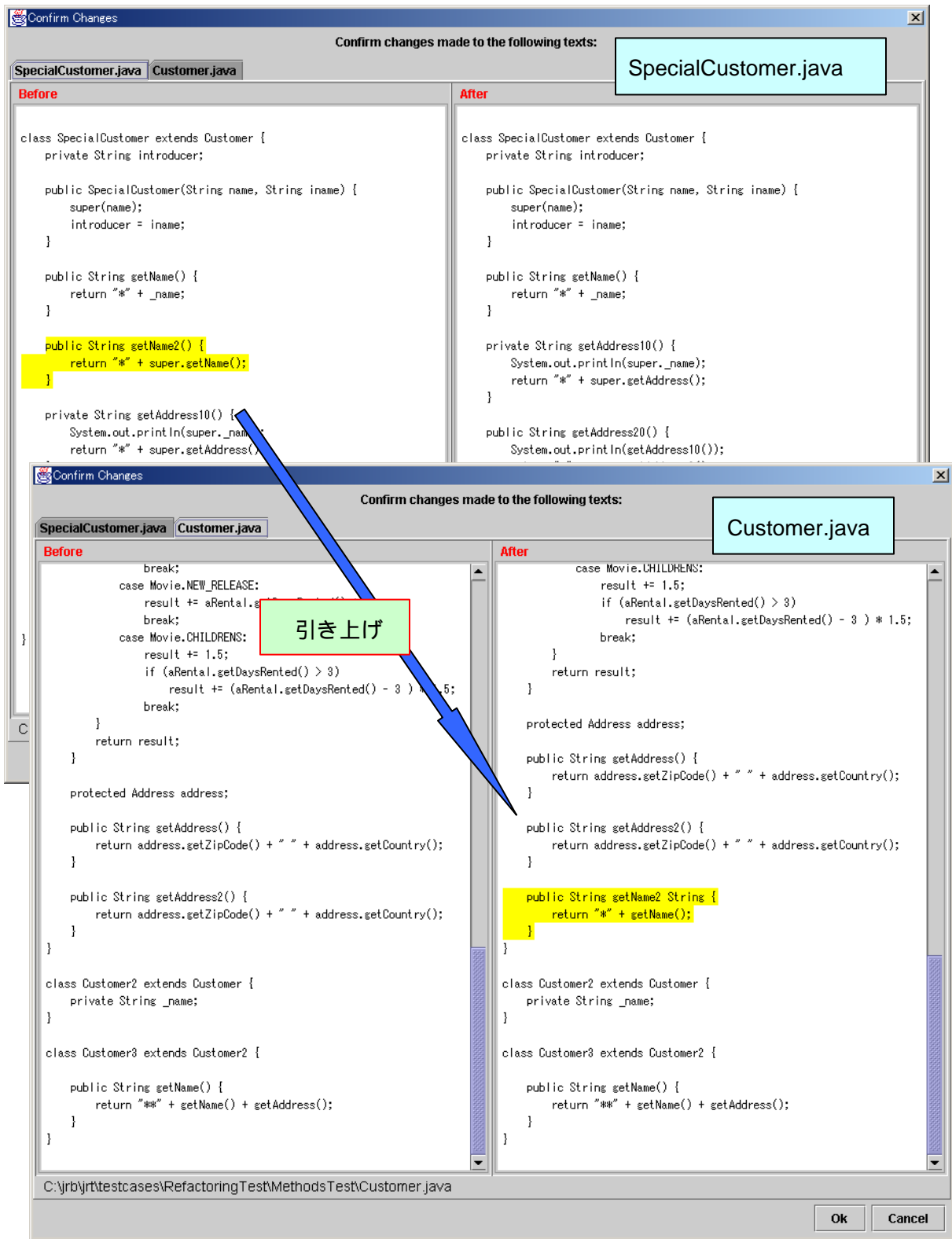
指定されたメソッドを直属のスーパークラスに移動する。手順は以下の通りである。

- (i) 引き上げ対象メソッドを所有するクラスのスーパークラスが引き上げ先クラスとして表示される。引き上げを実行する場合は、Ok ボタンを押す。引き上げを中止したい場合は、Cancel を押す。



- (ii) 引き上げメソッドが元のクラス(引き上げ先クラスのサブクラス)のメソッドを呼び出している場合は、呼び出されているメソッドに対応する抽象メソッドが引き上げ先に生成される。

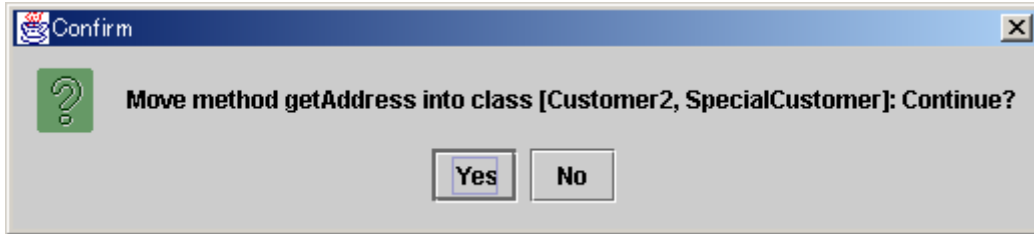
SpecialCustomer.java ファイルの SpecialCustomer クラスの getName2 メソッドを引き上げた際の実行結果を示す。本例では、getName2 メソッドは、Customer.java の Customer クラスに引き上げられる。



4.2.5 メソッドの引き下げ(Push Down Method)

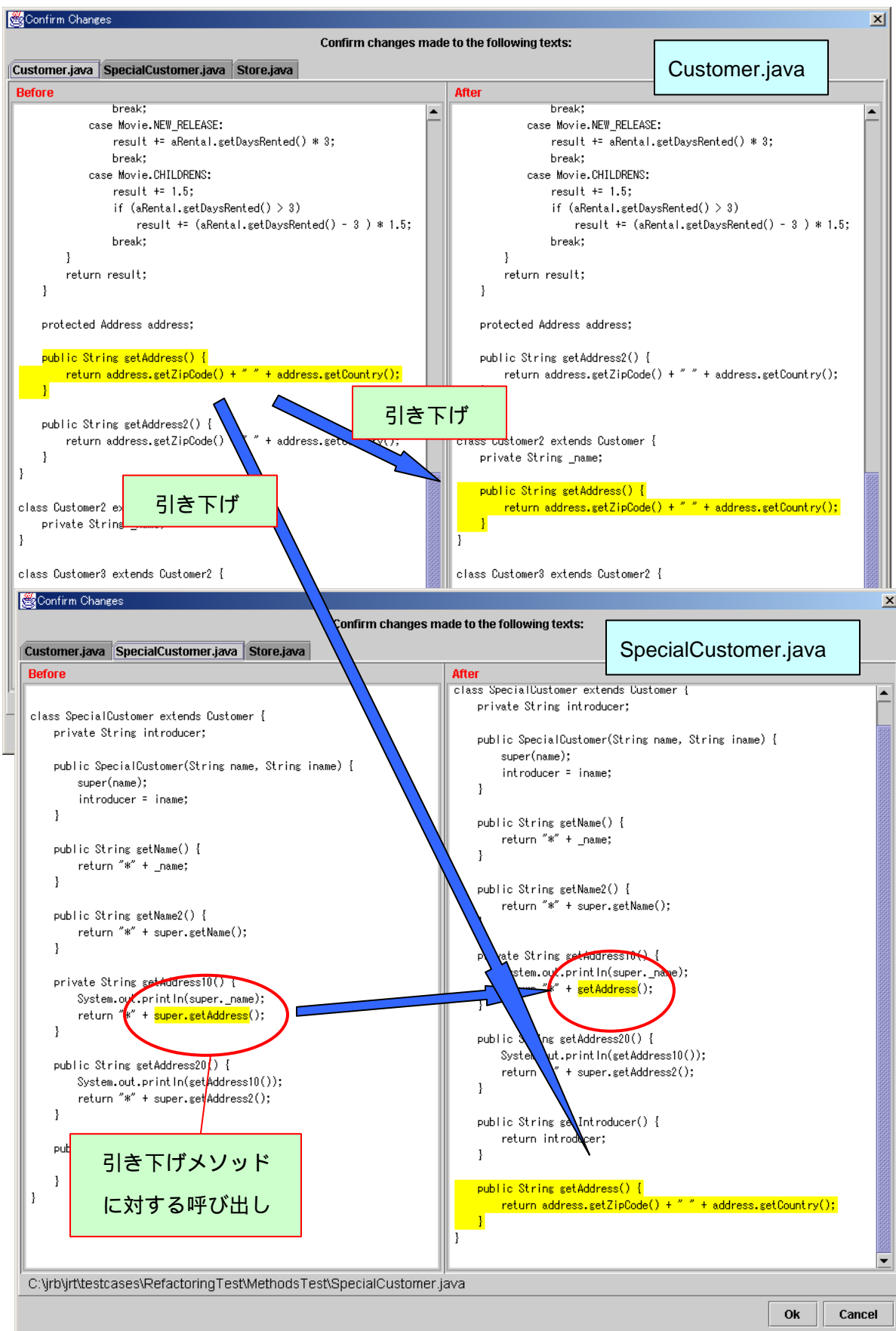
指定されたメソッドを直下のサブクラスに移動する。手順は以下の通りである。

- (i) 引き下げ対象メソッドを所有するクラスのサブクラスが引き下げ先クラスとして表示される。引き下げを実行する場合は、Ok ボタンを押す。引き下げを中止したい場合は、Cancel を押す。



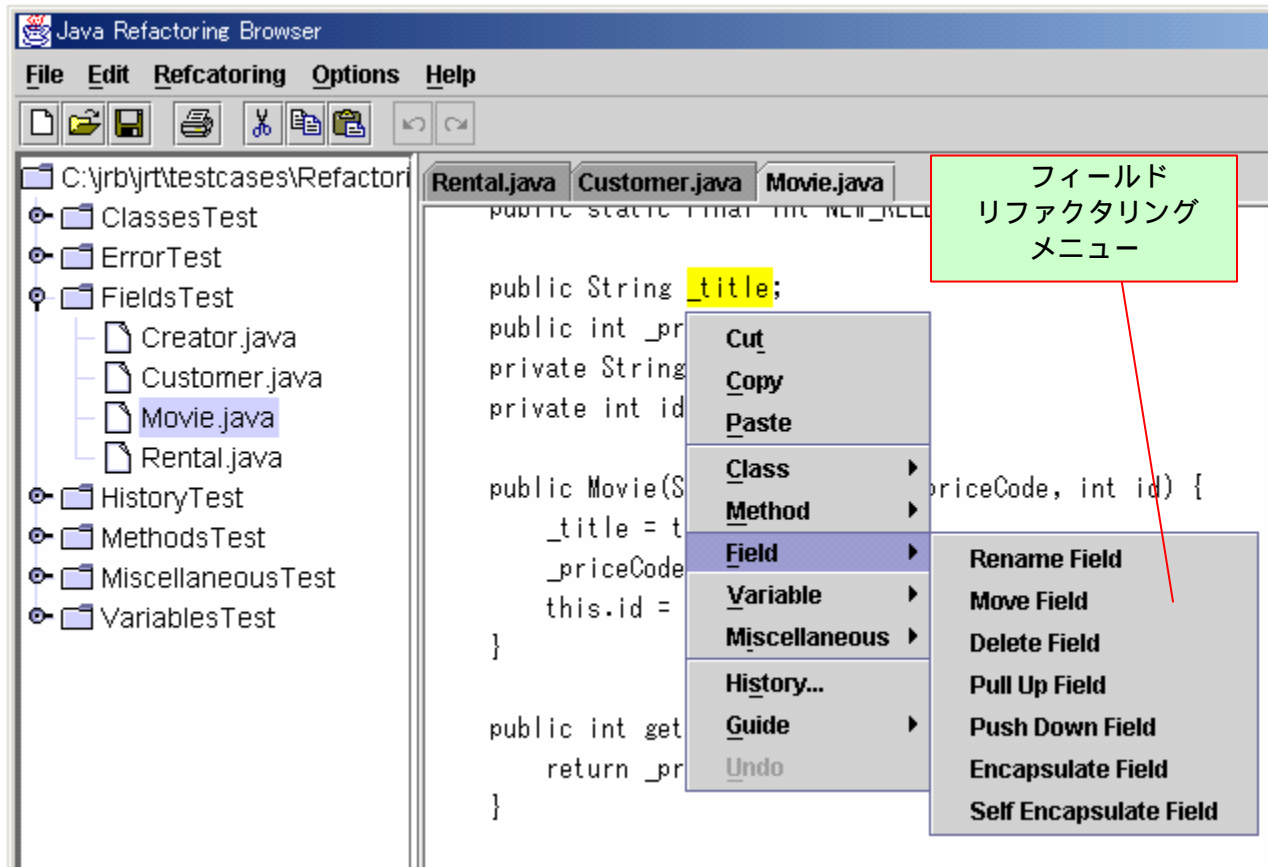
- (ii) 引き下げメソッドが元のクラ（引き下げ先クラスのスーパークラス）の private フィールドを参照している場合、そのフィールドの修飾子が protected に変更される。さらに、引き下げメソッドを呼び出していたクラスが探索ディレクトリ内に存在する場合、そのソースコードも確認画面に表示される。

Customer.java ファイルの Customer クラスの getAddress メソッドを引き下げた際の実行結果を示す。本例では、getAddress メソッドは、Customer.java の Customer2 クラスと SpecialCustomer.java ファイルの SpecialCustomer クラスに引き下げられる。さらに、本例では、Store クラスで引き下げメソッド getAddress を呼び出していたため、Store.java ファイルも同時に確認画面に表示されている（確認画面にタブが存在する）。



4.3 Fieldメニュー（フィールドに関するリファクタリング）

フィールドに関するサブメニューは，ソースコード編集画面においてフィールド宣言あるいはフィールド変数を選択した状態（どこか一箇所が良い）で表示され，実行可能となる．JRBでは，次に示す7つのフィールドリファクタリング操作を提供する．



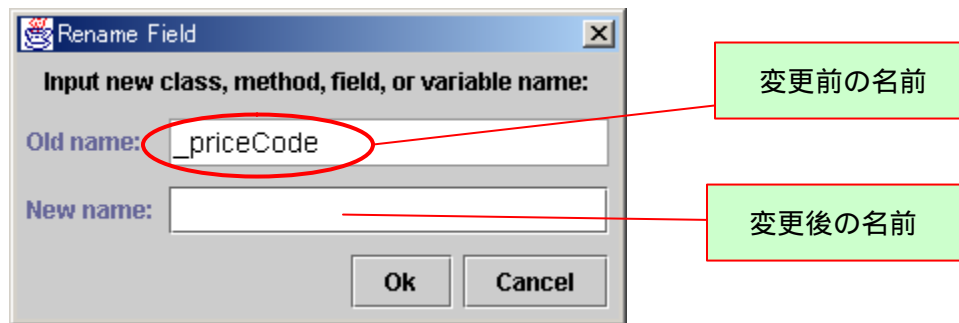
- (1) フィールド名の変更(Rename Field)
- (2) フィールドの移動(Move Field)
- (3) フィールドの削除>Delete Field)
- (4) フィールドの引き上げ(Pull Up Field)
- (5) フィールドの引き下げ(Push Down Field)
- (6) カプセル化(Encapsulate Field)
- (7) 自己カプセル化(Self Encapsulate Field)

それぞれのリファクタリング操作について，その実行手順を4.3.1～4.3.7に示す．

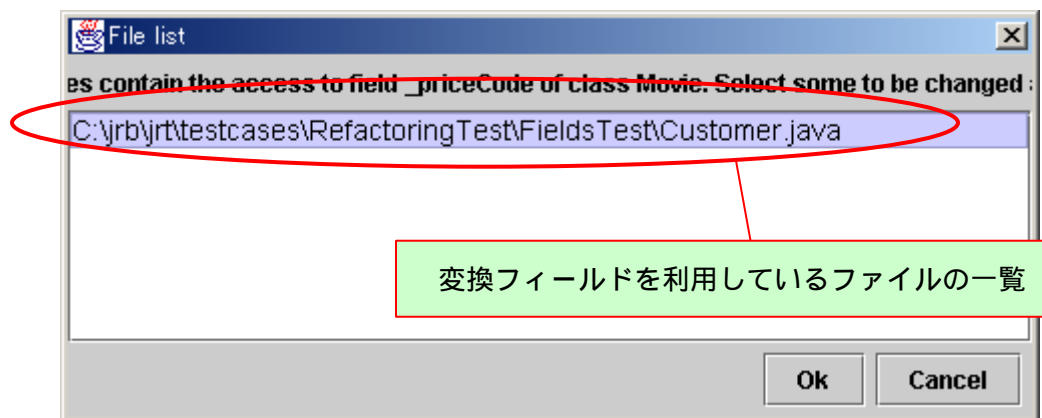
4.3.1 フィールド名の変更(Rename Field)

指定されたフィールドの名前を変更する。手順は以下の通りである。

- (i) 新規フィールド名入力ダイアログが表示されるので、New Name に新しいフィールドの名前を入力し、Ok ボタンを押す。変更を中止したい場合は、Cancel ボタンを押す。Old Name には変更前の名前が自動的に表示される（この文字列は変更不可能）

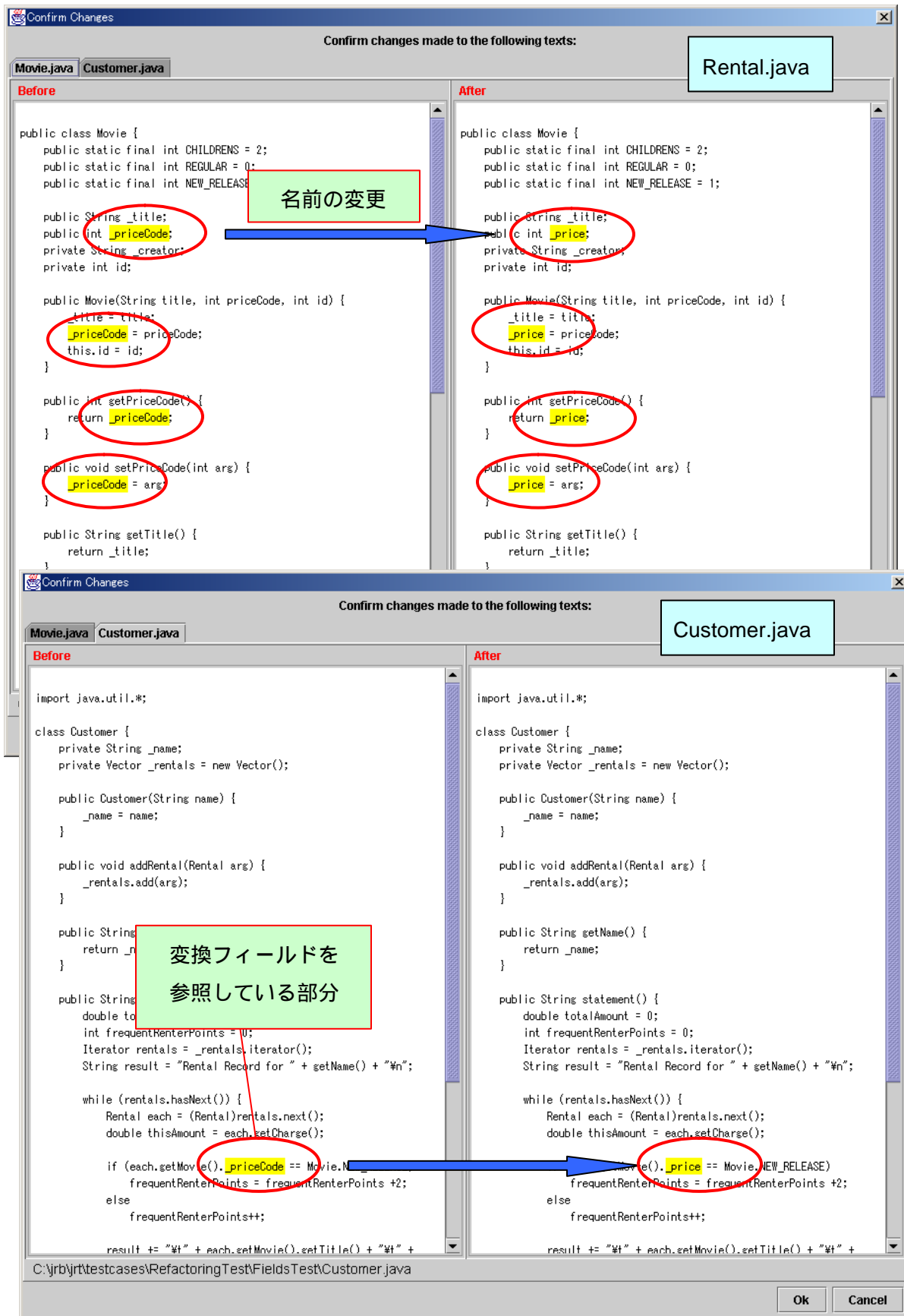


- (ii) 名前を変更するフィールドを利用しているクラスが探索ディレクトリ内に見つかった場合、そのクラスを含むファイルの一覧が表示される。



この一覧より、フィールド名（参照名）を同時に変換するファイルを選択し、Ok ボタンを押す。上図は Customer.java ファイルを選択した例である。

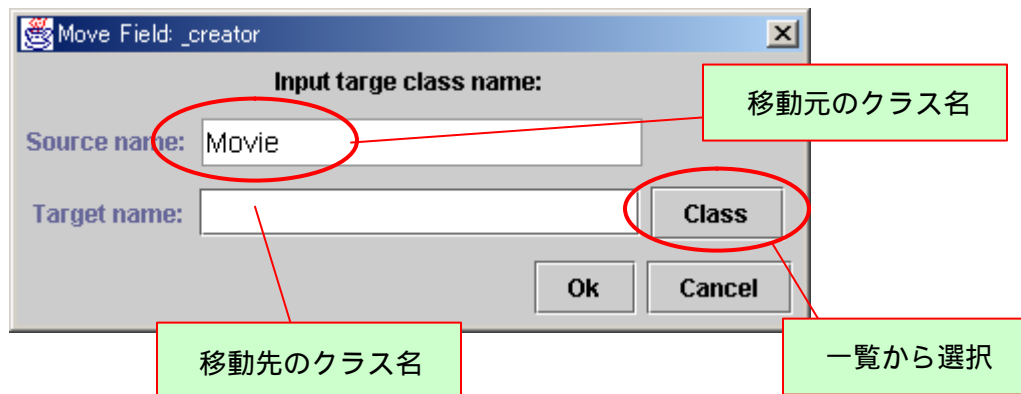
Rental.java ファイルの Rental クラスのフィールド _priceCode を _price に変更した際の実行結果を以下に示す。



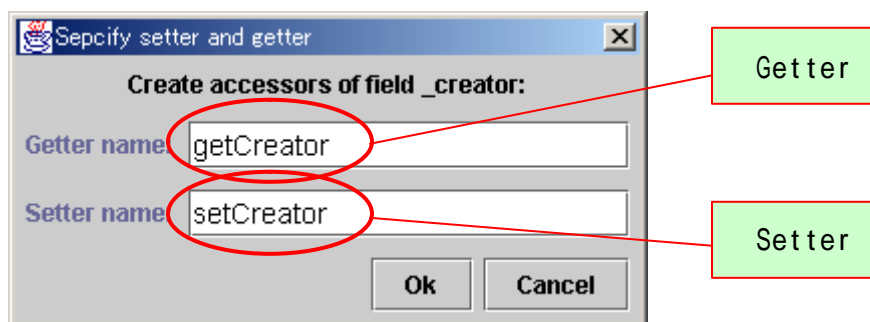
4.3.2 フィールドの移動(Move Field)

指定されたフィールドを移動する。手順は以下の通りである。

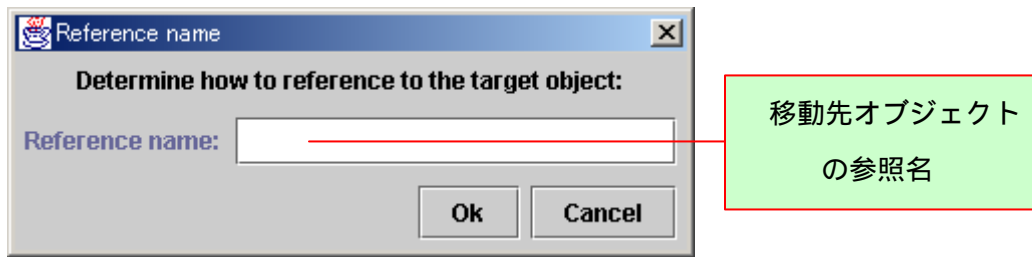
- (i) 移動先クラスの入力ダイアログが表示されるので、Target name に移動先のクラス名を入力し、Ok ボタンを押す。Source Name には移動前のクラス名が自動的に表示される（この文字列は変更不可能）。クラス名の指定形式は、4.1.3 と同様である。クラス名だけを指定した場合、移動フィールドを所有するクラスと同じパッケージに存在するクラスが移動先クラスとして探索される。



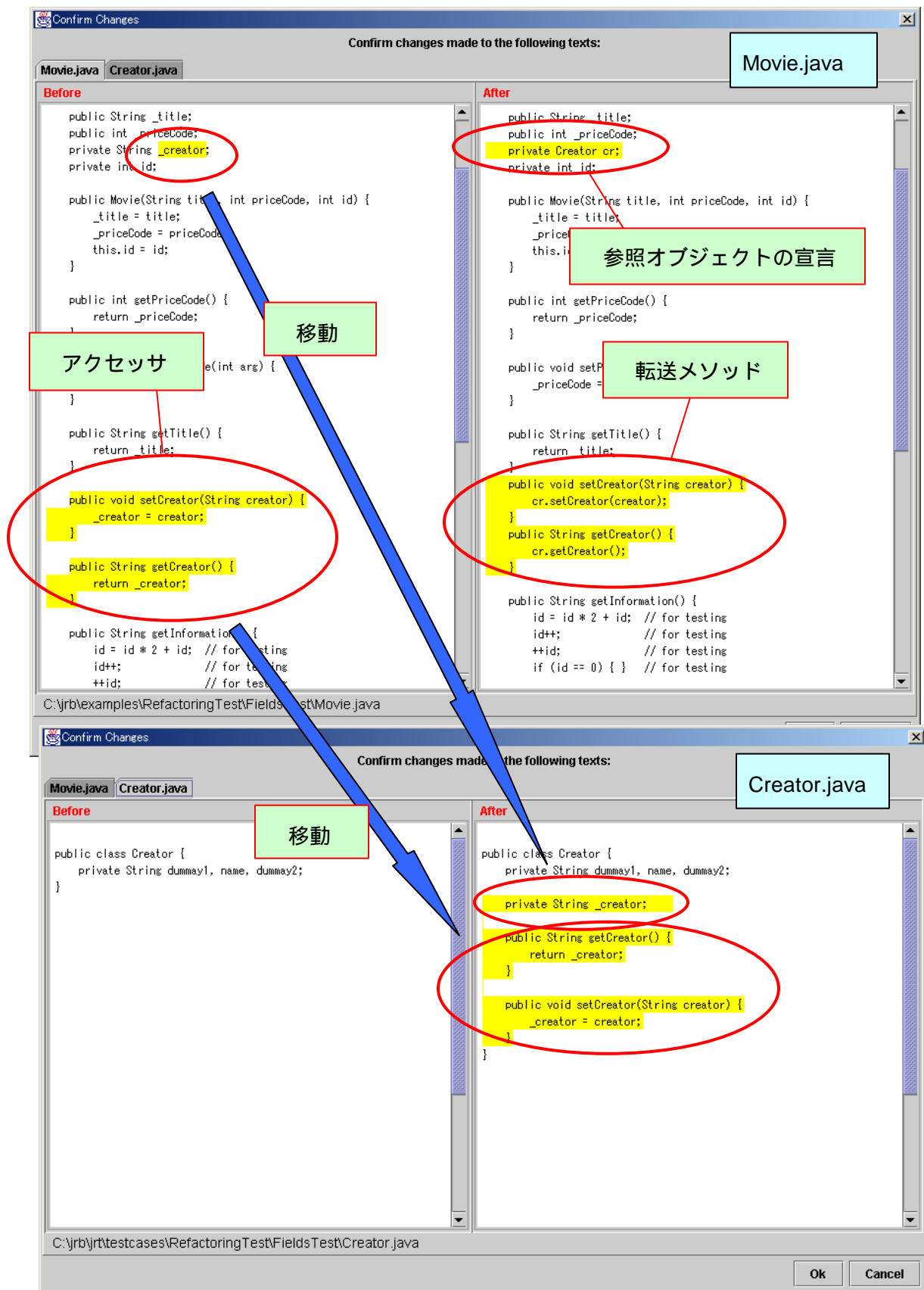
- (ii) フィールドと同時に移動するアクセッサ（Getter および Setter）を指定するダイアログが表示される。アクセッサの名前は、デフォルトで「get + フィールド名」「set + フィールド名」である（フィールド名の先頭 1 文字は大文字に変換される）。指定されたアクセッサが存在しない場合、フィールドの移動は不可能と判断されるため、その場合は事前に、後述する「4.3.6 カプセル化」によりアクセッサを生成しておく。



- (iii) 移送先クラスに対する参照オブジェクト名の入力するダイアログが表示される。参照オブジェクト名が決定されている場合はその名前を入力し、Ok ボタンを押す。参照オブジェクト名が未決定の場合は、Cancel ボタンを押すことで、先に進むことができる。この場合、アクセッサに対する転送メソッドは生成されない。



Movie.java ファイルの Movie クラスのフィールド _creator を Creator.java ファイルの Creator クラスに移動した際の実行結果を以下に示す。

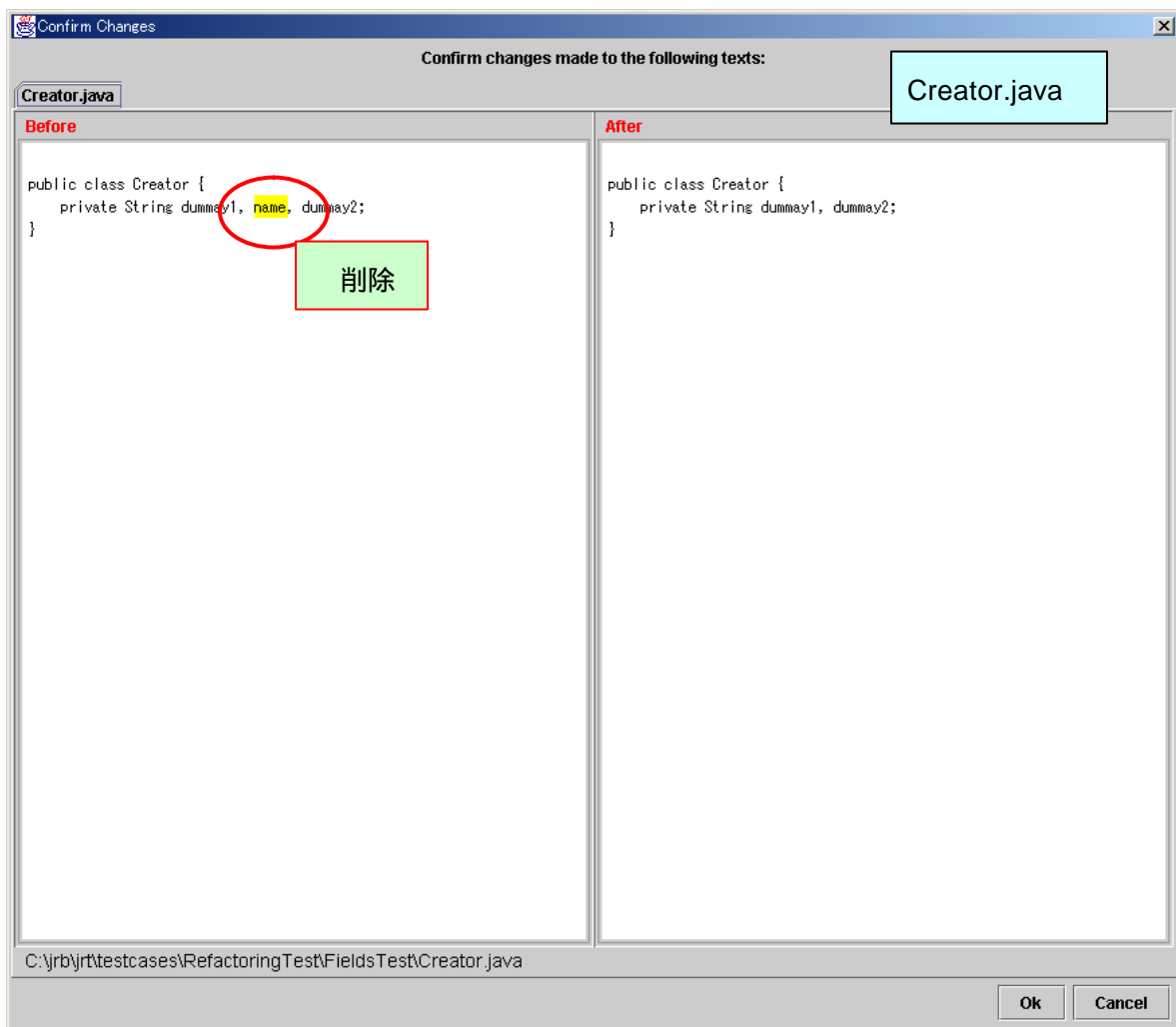


4.3.3 フィールドの削除(Delete Field)

指定されたフィールドを削除する．手順は以下の通りである．

- (i) 削除するフィールドを利用しているメソッドあるいはフィールドが Root directory 以下のディレクトリに対して検索され ,そのようなメソッドあるいはフィールドが存在しない場合のみ削除が実行される .削除フィールドを利用しているメソッドあるいはフィールドが存在する場合は ,警告ダイアログが表示され ,削除は中止される .

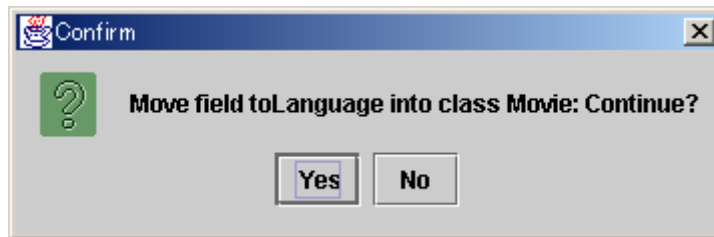
Creator.java ファイルの Creator クラスの name フィールドを削除した際の実行結果を以下に示す .



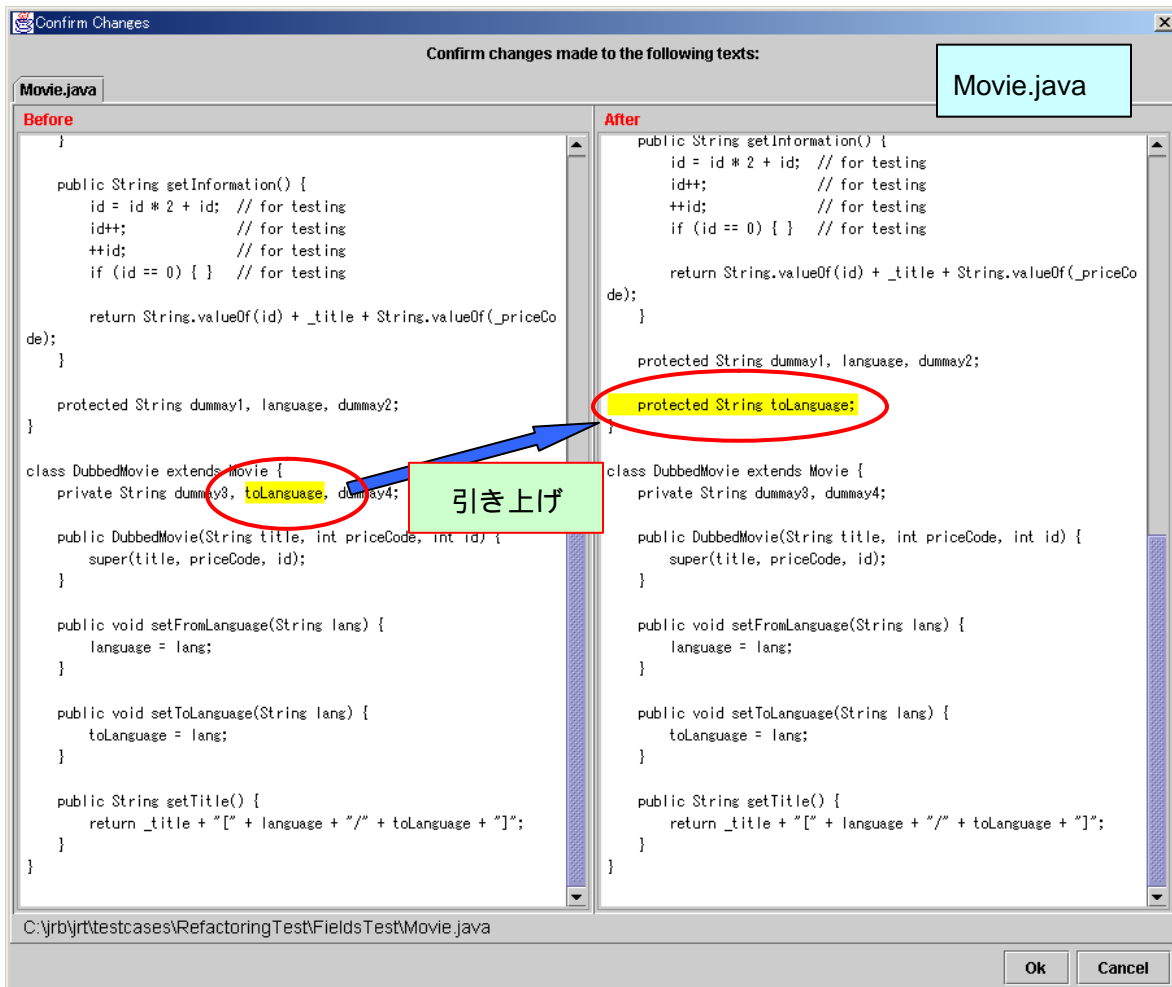
4.3.4 フィールドの引き上げ(Pull Up Field)

指定されたフィールドを直属のスーパークラスに移動する。手順は以下の通りである。

- (i) 引き上げ対象フィールドを所有するクラスのスーパークラスが引き上げ先クラスとして表示される。引き上げを実行する場合は、Ok ボタンを押す。引き上げを中止したい場合は、Cancel を押す。引き上げフィールドが元のクラス（引き上げ先クラスのサブクラス）で参照されている場合、引き上げフィールドの修飾子は protected となる。



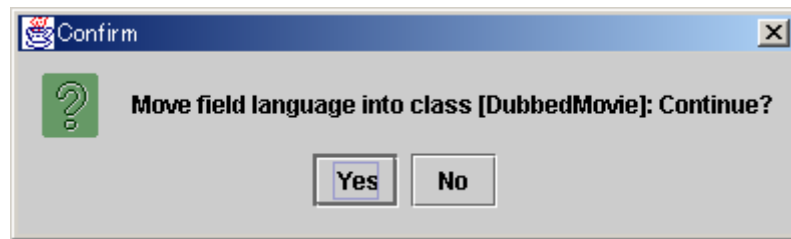
Movie.java ファイルの DebbedMovie クラスの toLanguage フィールドを引き上げた際の実行結果を示す。本例では、toLanguage は、Movie.java ファイルの Movie クラスに引き上げられる。



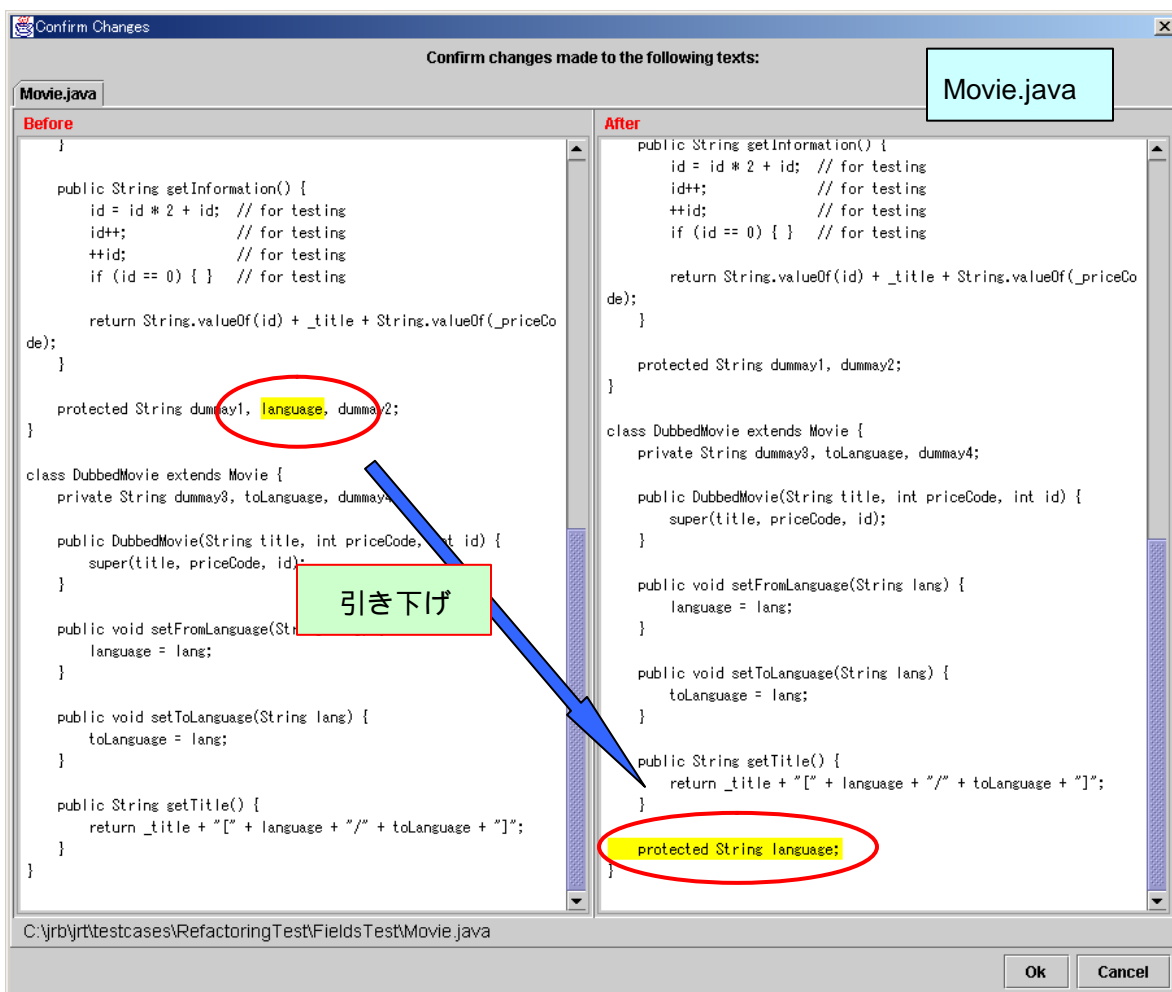
4.3.5 フィールドの引き下げ(Push Down Field)

指定されたフィールドを直下のサブクラスに移動する．手順は以下の通りである．

- (i) 引き下げ対象フィールドを所有するクラスのサブクラスが引き下げ先クラスとして表示される．引き下げを実行する場合は，Ok ボタンを押す．引き下げを中止したい場合は，Cancel を押す．



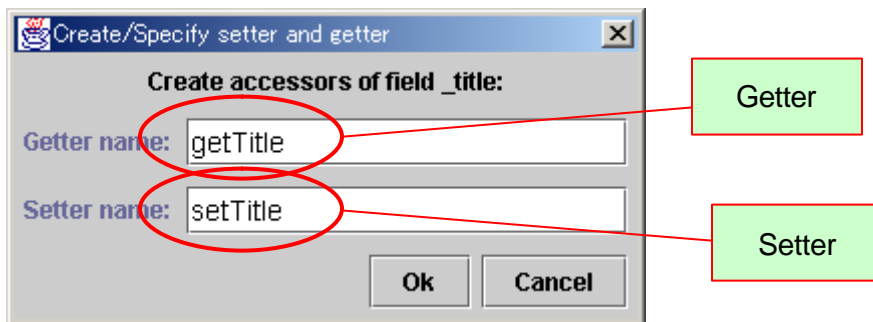
Movie.java ファイルの Movie クラスの language フィールドを引き下げた際の実行結果を示す．本例では，language は，DebbbedMovie.java ファイルの DebbbedMovie クラスに引き下げられる．



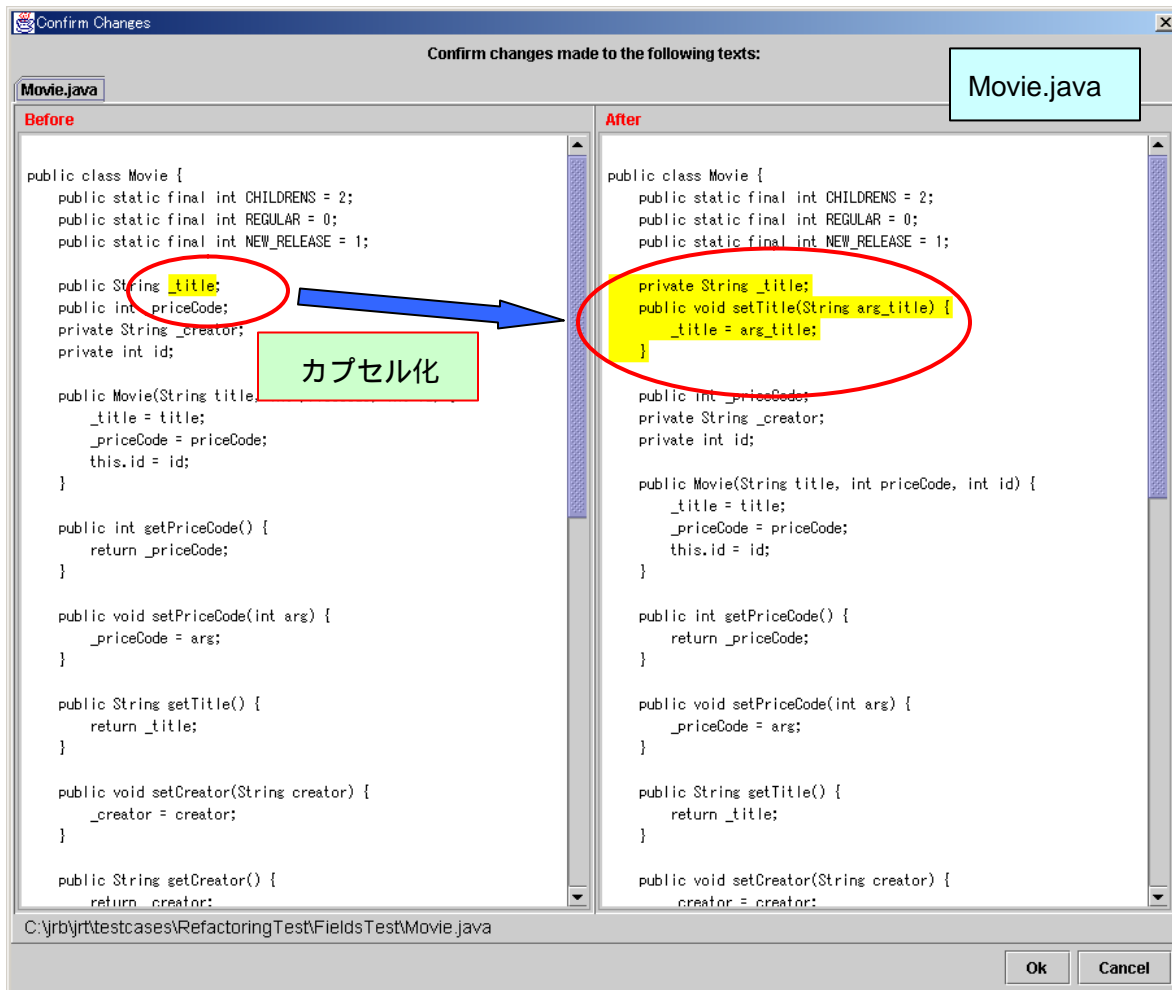
4.3.6 カプセル化(Encapsulate Field)

指定された公開(public)フィールドを非公開(private)フィールドに変更し,アクセッサを提供する。手順は以下の通りである。

- (i) カプセル化対象フィールドのアクセッサ(Getter および Setter)を入力するダイアログが表示される。アクセッサのデフォルトの名前は, 4.3.2 と同様である。指定されたアクセッサと同名のメソッドが既に存在する場合, そのメソッドはアクセッサとして扱われる。カプセル化を実行する場合は, アクセッサの入力後に Ok ボタンを押す。カプセル化を中止したい場合は, Cancel を押す。



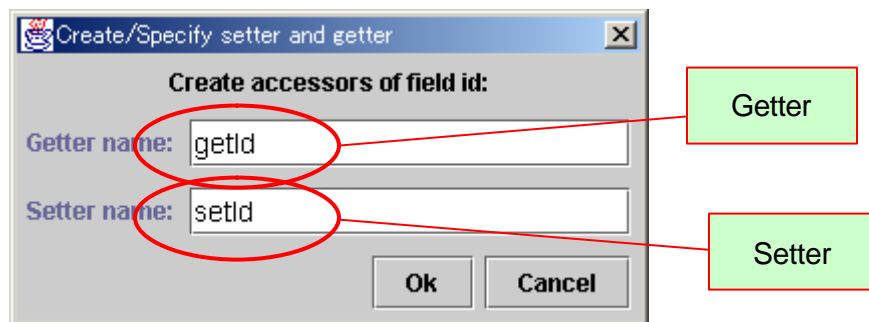
Movie.java ファイルの Movie クラスの_title フィールドをカプセル化した際の実行結果を示す。本例では, もとの Movie クラスにすでに Getter (getTitle メソッド) が存在していたため ,Setter (setTitle メソッド) のみが生成されている。



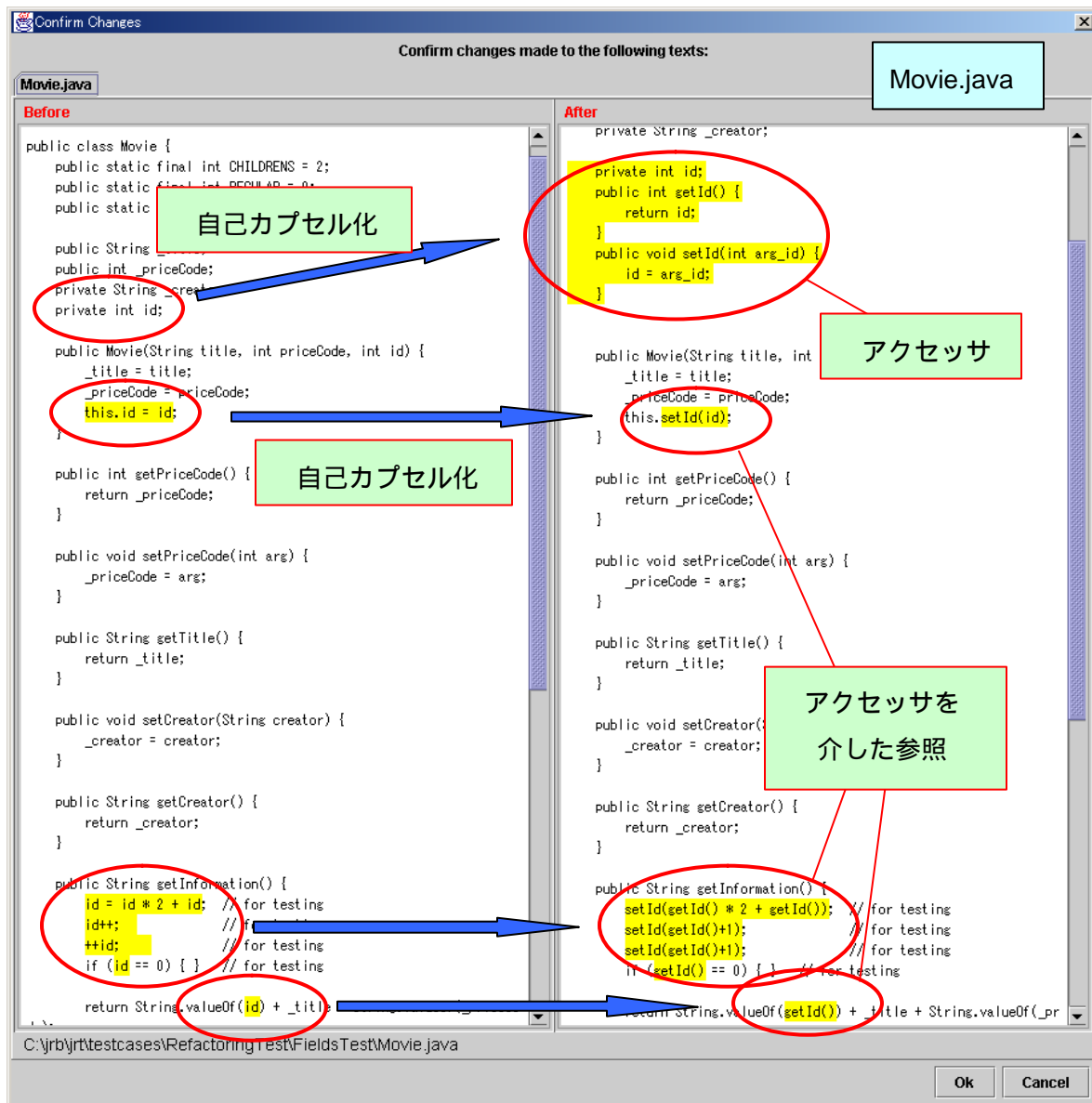
4.3.6 自己カプセル化(Self Encapsulate Field)

指定されたフィールドに直接アクセスしている箇所を，アクセッサを介したアクセスに変更する．手順は以下の通りである．

- (i) 自己カプセル化対象フィールドのアクセッサ（Getter および Setter）を入力するダイアログが表示される．アクセッサのデフォルトの名前は，4.3.2 と同様である．指定されたアクセッサが存在しない場合，アクセッサが生成される．自己カプセル化を実行する場合は，アクセッサの入力後に Ok ボタンを押す．自己カプセル化を中止したい場合は，Cancel を押す．

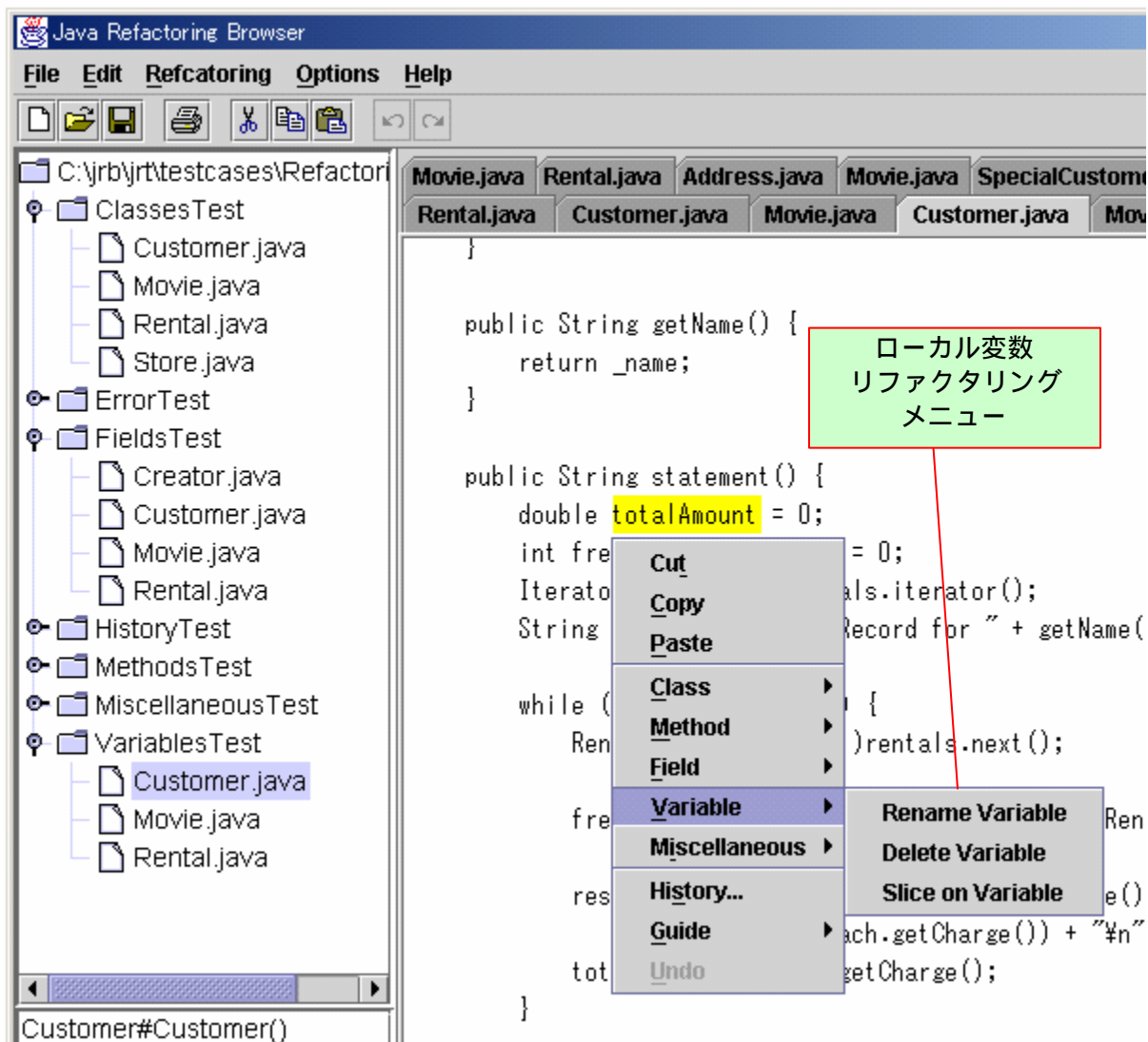


Movie.java ファイルの Movie クラスの id フィールドを自己カプセル化した際の実行結果を示す．



4.4 Variable メニュー（ローカル変数に関するリファクタリング）

ローカル変数（メソッドのパラメータを含む）に関するサブメニューは，ソースコード編集画面においてローカル変数あるいはメソッドのパラメータを選択した状態（どこか一箇所が良い）で表示され，実行可能となる．JRB では，次に示す 3 つのローカル変数リファクタリング操作を提供する．



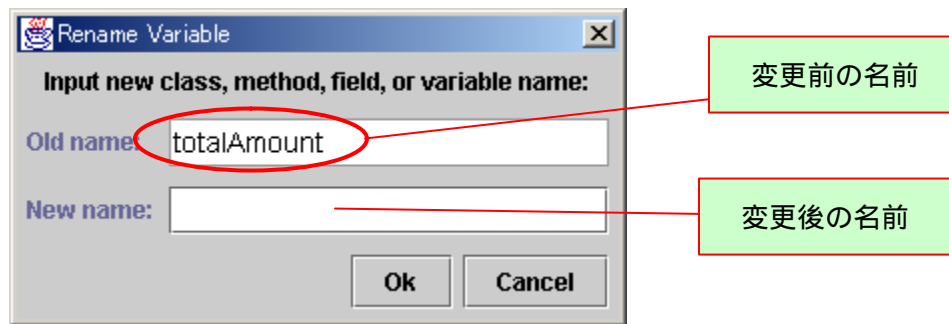
- (1) 変数名の変更(Rename Variable)
- (2) 変数の削除>Delete Variable)
- (3) スライスの抽出(Slice On Variable)

それぞれのリファクタリング操作について，その実行手順を 4.4.1 ～ 4.4.3 に示す．

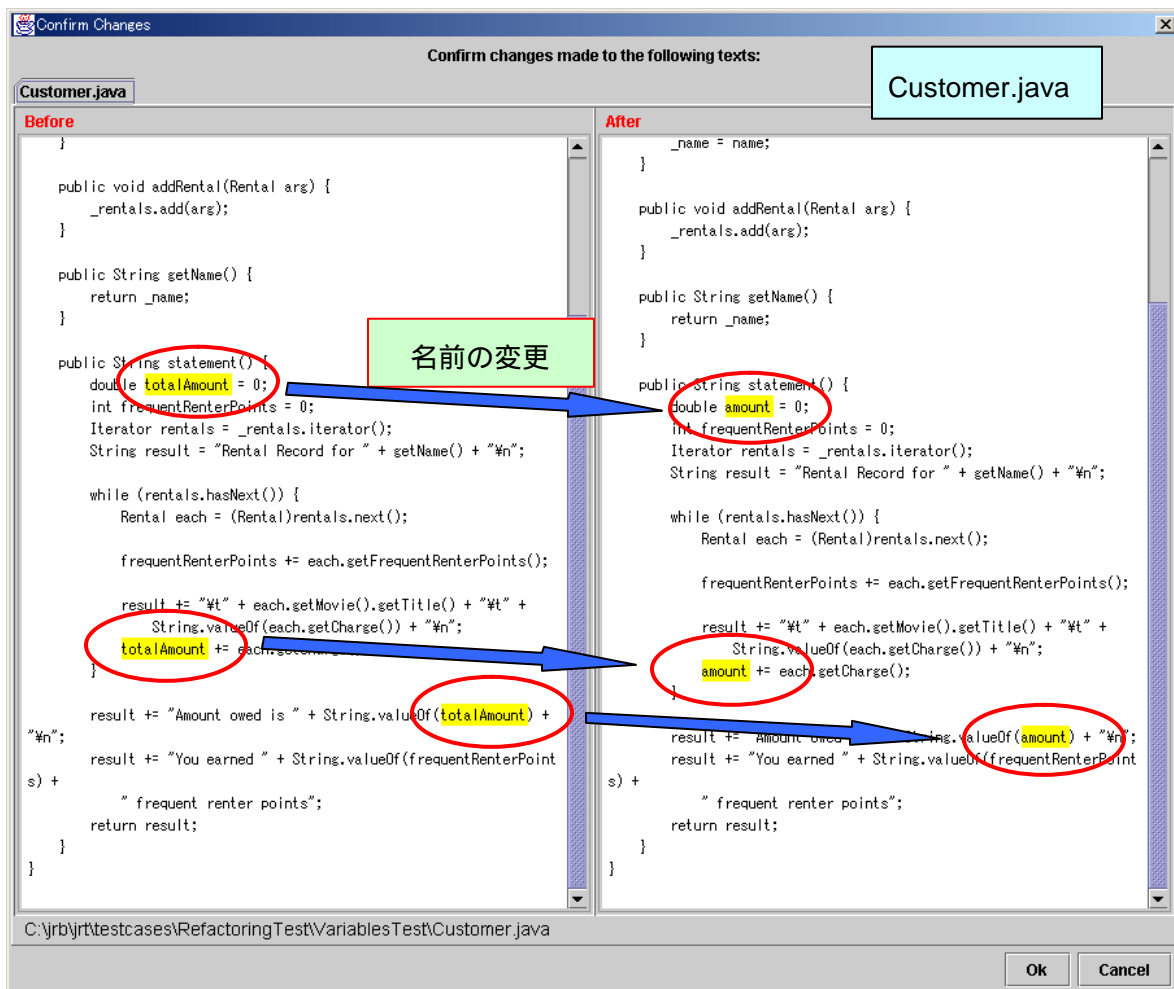
4.4.1 変数名の変更(Rename Variable)

指定されたローカル変数の名前を変更する．手順は以下の通りである．

- (i) 新規変数名入力ダイアログが表示されるので，New Name に新しい変数の名前を入力し，Ok ボタンを押す．変更を中止したい場合は，Cancel ボタンを押す．Old Name には変更前の名前が自動的に表示される（この文字列は変更不可能）



Customer.java ファイルの Customer クラスの totalAmount 変数の名前を amount に変更した際の実行結果を以下に示す．

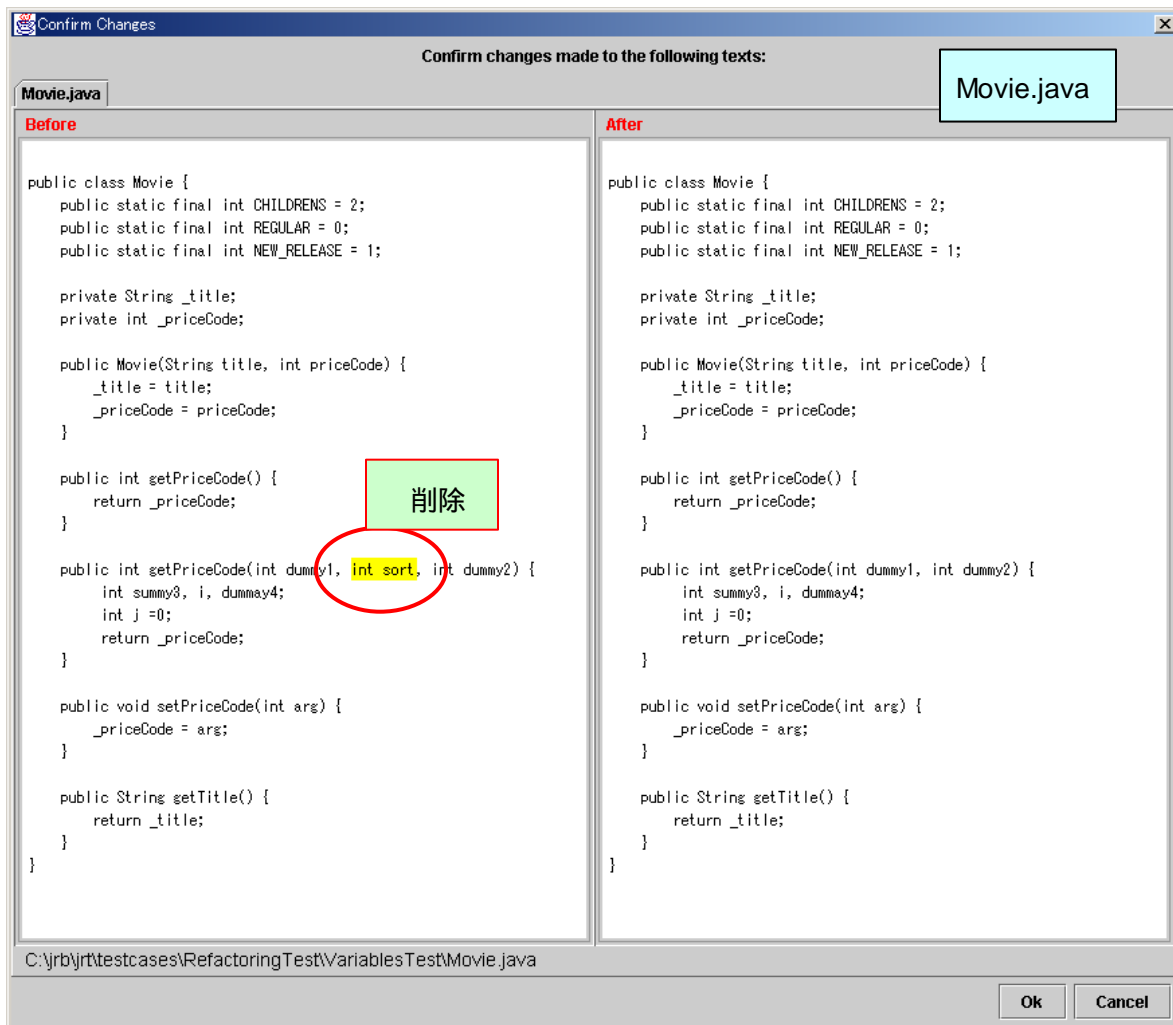


4.4.2 変数の削除(Delete Variable)

指定されたローカル変数を削除する。手順は以下の通りである。

- (i) 削除するローカル変数が、その変数が存在するメソッド内で宣言されているだけ、あるいは、その変数の値が一度だけ定義され参照がない場合のみ、削除が実行される。それ以外の場合は、警告ダイアログが表示され、削除は中止される。

Movie.java ファイル内の Movie クラスの sort 変数を削除した際の実行結果を以下に示す。



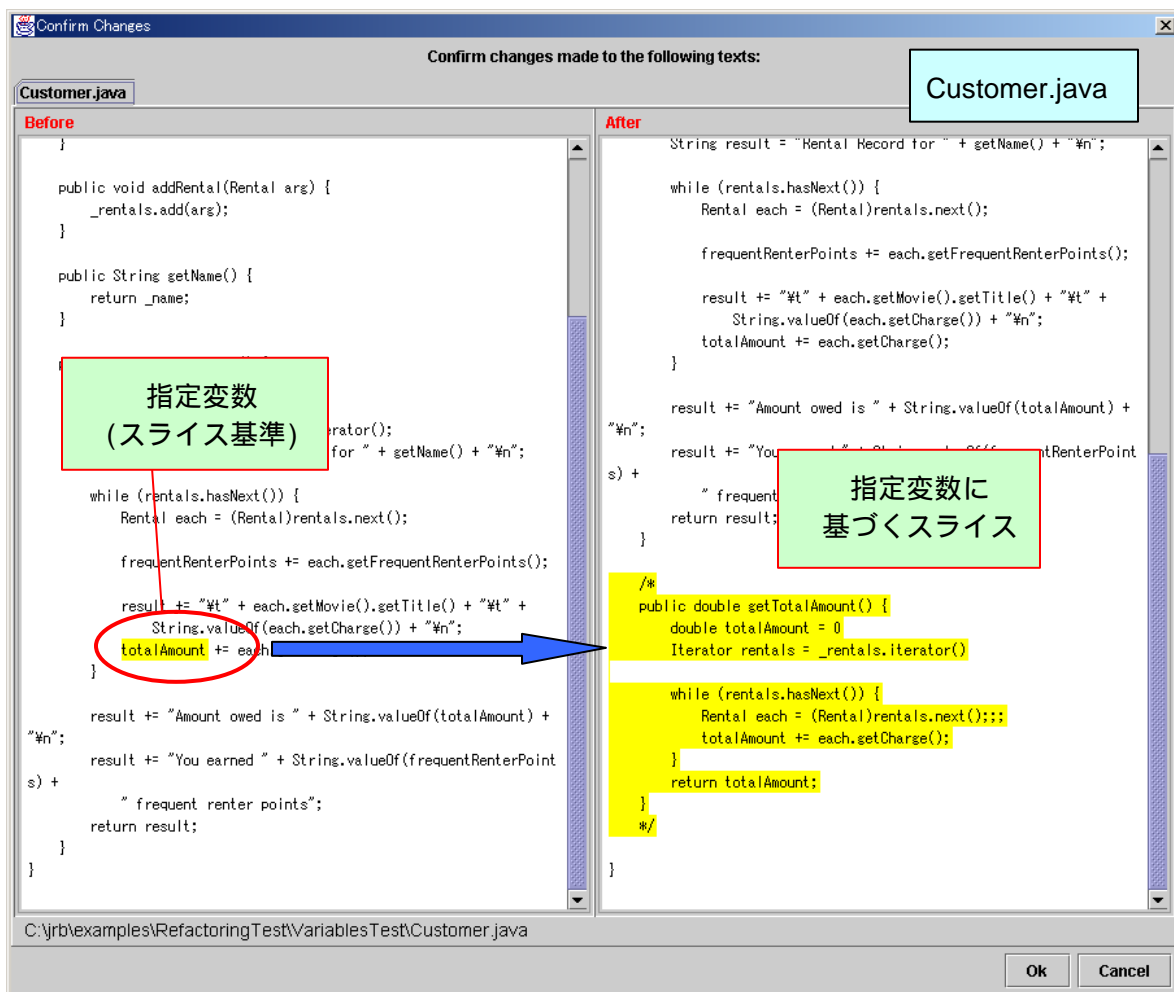
4.4.3 スライスの抽出(Slice on Variable)

指定されたローカル変数に基づくスライスを抽出し、メソッドとして整形する。手順は以下の通りである。スライスとは、指定したローカル変数の値を計算するために必要な文をもとのメソッド内部から集めたものである。

- (i) スライスによるメソッド抽出では、完全に実行（あるいはコンパイル）可能なソースコードを出力する保証がないため、生成したコードはコメントとなることの確認ダイアログが表示される。

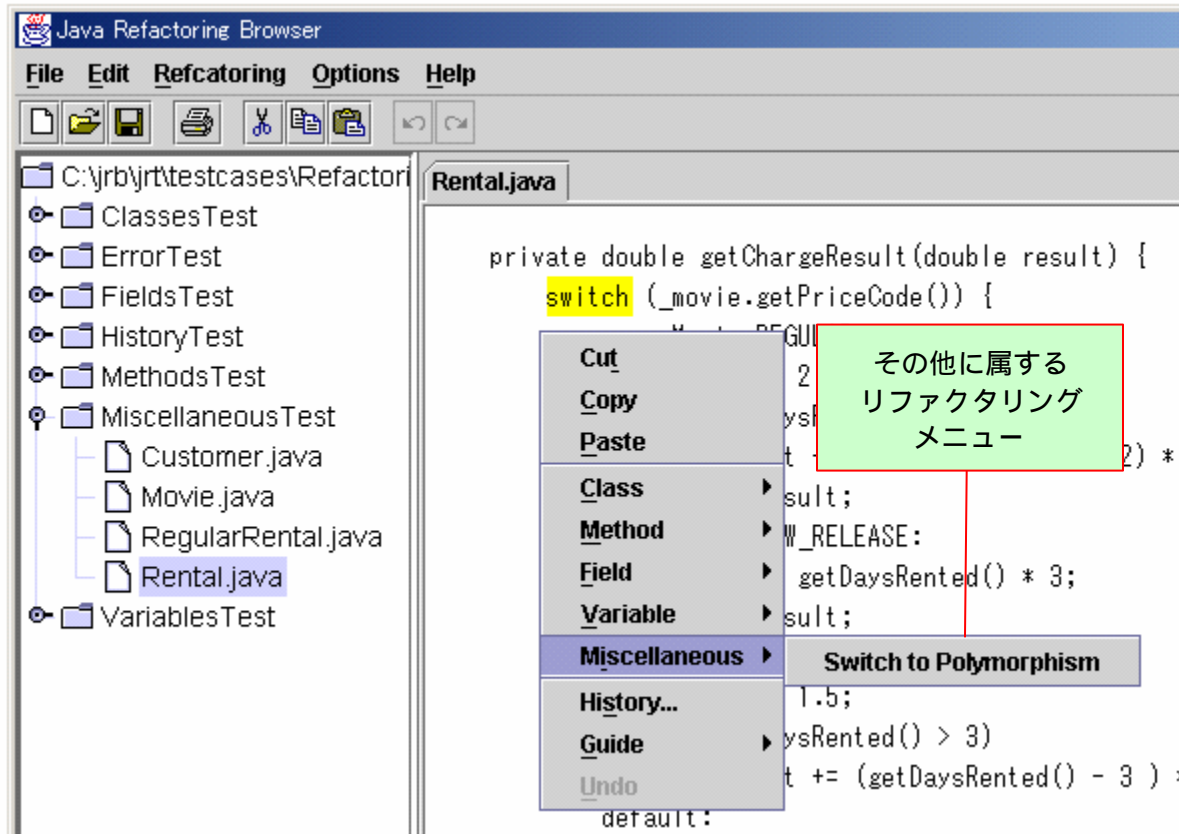


Customer.java ファイルの Customer クラスの totalAmount 変数に基づきスライスを抽出した際の実行結果を以下に示す。



4.5 Miscellaneous メニュー（その他のリファクタリング）

4.1 ~ 4.4 に分類できないリファクタリングがこのメニューに属する。現在 switch 文に関するリファクタリングがサブメニューとして用意されている。



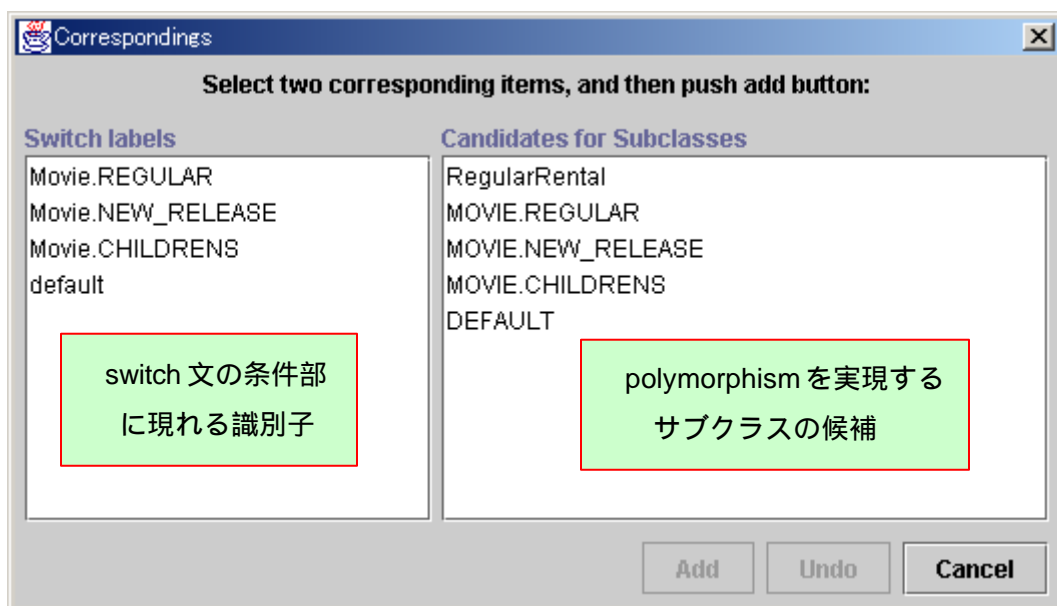
(1) polymorphism による条件分岐(switch 文)の置き換え(Switch to Polymorphism)

このリファクタリング操作について、その実行手順を 4.5.1 に示す。

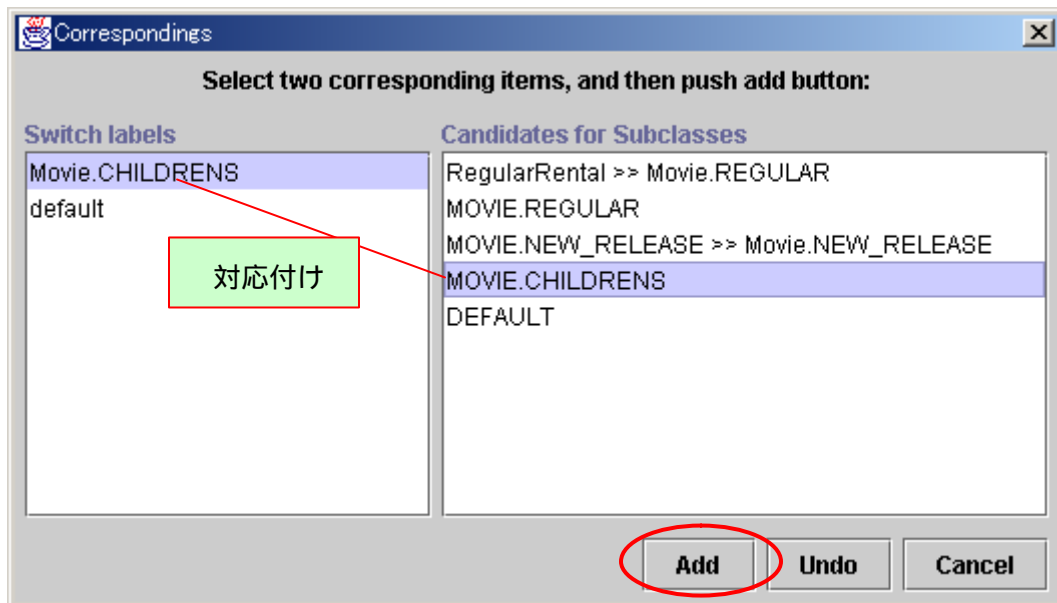
4.5.1 polymorphismによる条件分岐(switch文)の置き換え(Switch to Polymorphism)

このリファクタリングメニューは、ソースコード編集画面において switch 文 (switch 文字列) を選択した状態で表示され、実行可能となる。switch 文の各条件部に対応するアクション部をサブクラスのオーバーライドメソッドとして移動する。もとのメソッドは、抽象メソッドとなり、polymorphism を実現する。手順は以下の通りである。

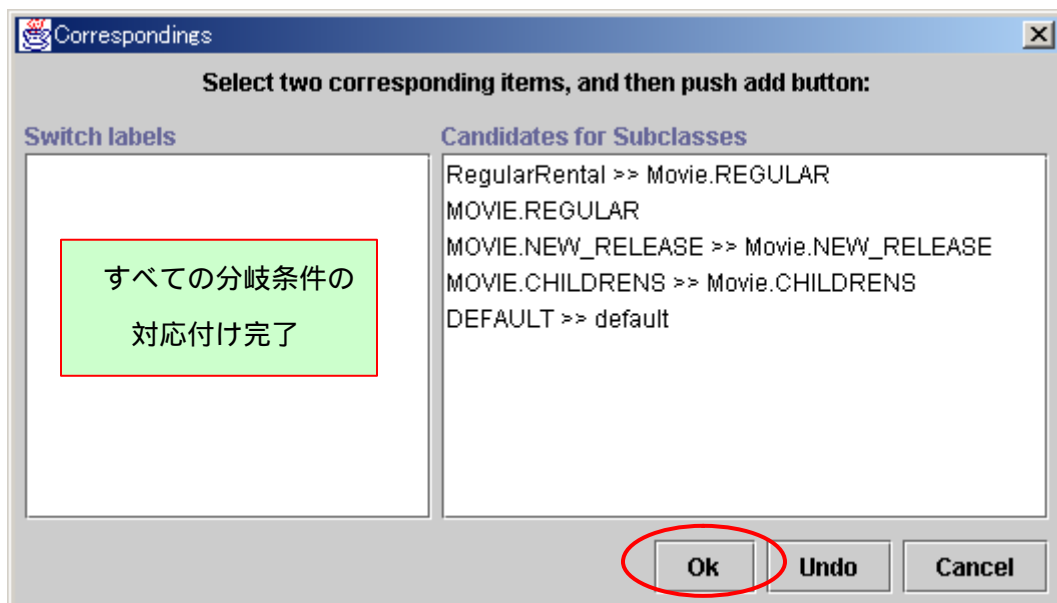
- (i) switch 文の条件部に対応するサブクラスを指定するためのダイアログが表示される。左側には、switch 文の分岐条件が列挙される。また、右側には、もとのメソッドのサブクラスと新規作成クラス (分岐条件部の識別子を大文字に変換した名前を持つクラス) が、polymorphism を実現するサブクラスの候補として列挙される。
たとえば、次のダイアログでは、左側の「Movie.REGULAR」「Movie.NEW_RELEASE」「Movie.CHILDRENS」「default」が、switch 文の分岐条件である。また、もとのクラス(Rental)には、サブクラス RegularRental のみが存在しており、その他のクラスは新規に生成される可能性があるクラスである。



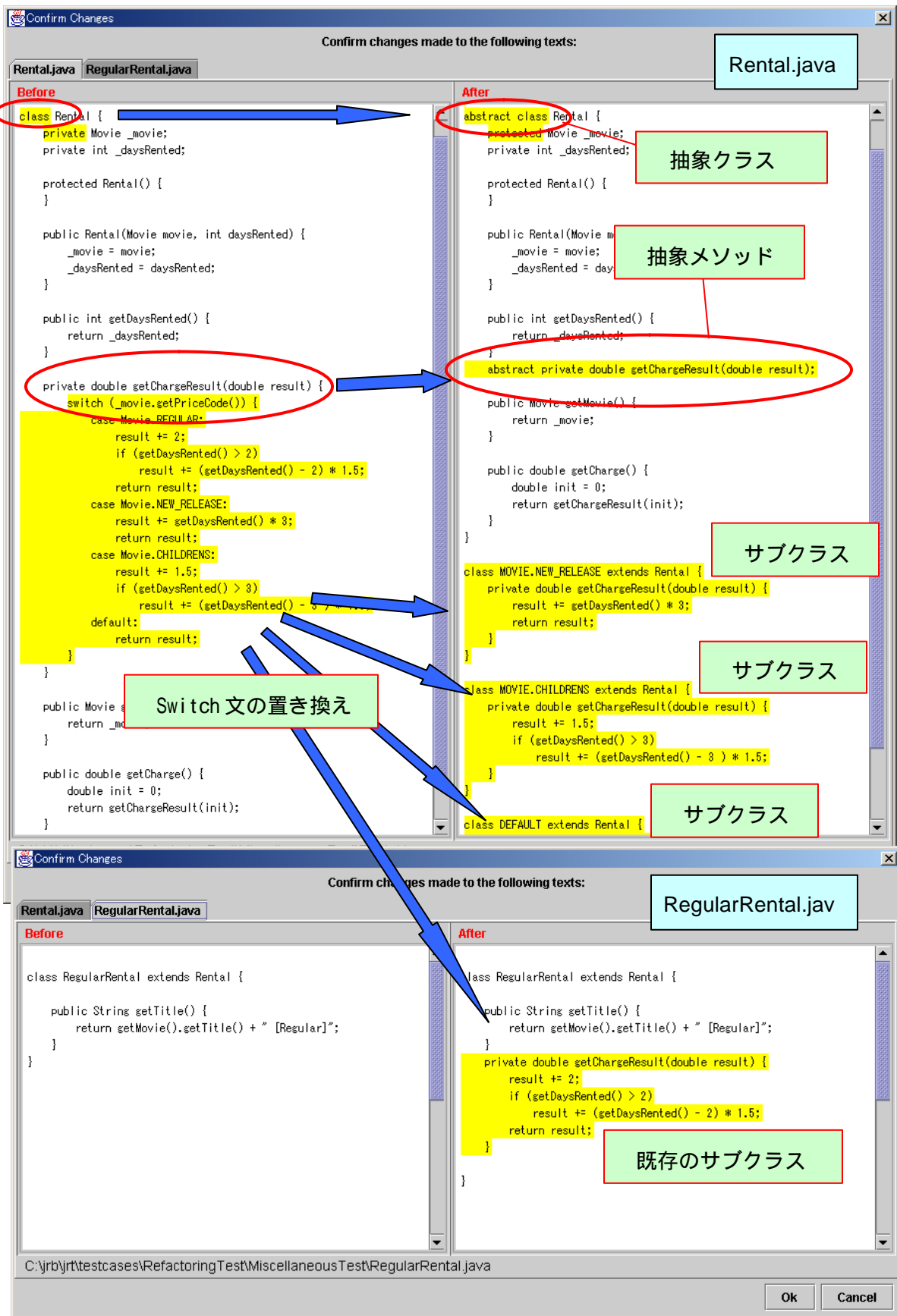
- (ii) 各分岐条件とサブクラスの対応を指定する。対になる 2 つの項目を選択し、Add ボタンを押すことで、対応付けが行われる。次の図は、「Movie.CHILDRENS」と「MOVIE.CHILDRENS」クラスの対応付けを行っている様子である。この図では、既に「Movie.REGULAR」と「RegularRental」クラス、「Movie.NEW_RELEASE」と「MOVIE.NEW_RELEASE」の対応付けが行われている。対応付けが終わった分岐条件は、右側に移動する。また、Undo ボタンにより、対応付けを取り消すことができる。



- (iii) すべての分岐条件に対するサブクラスが対応付けられると、Add ボタンが Ok ボタンに変化するので、switch 文の変換を実行する場合は、Ok ボタンを押す。分岐条件と対応付けされていないクラスに関しては新規に生成しない。この時点で変換を中止する場合は、Cancel ボタンを押す。



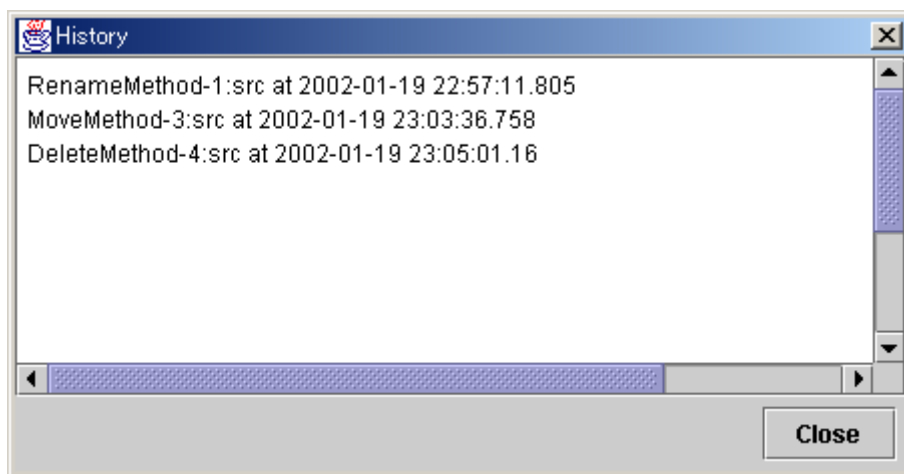
Rental.java ファイルの Rental クラスの switch 文を polymorphism に置き換えた際の実行結果を次に示す。



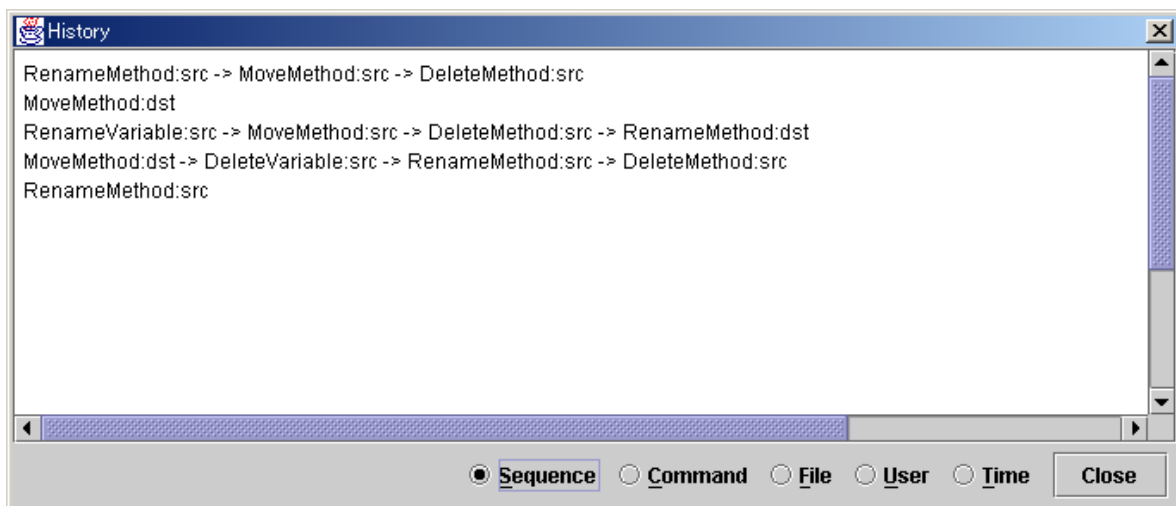
4.6 History メニュー（リファクタリング履歴の表示）

ソースコード編集画面の最前面のソースコードに対するリファクタリングの履歴を、それらの操作が行われた順序で表示する。この履歴は、ソースコードを保存した際に、全体の履歴に登録され、ソフトウェア終了後も残る。反対に、ソースコードを保存せずにファイルをクローズする、あるいは、保存せずにソフトウェアを終了した場合、これらの履歴は破棄される。

履歴の形式は、「リファクタリング操作名」+「-」+「識別番号」+「:」+「区分」+「操作時刻」である。識別番号は、同名のリファクタリング操作を区別するためにソフトウェア開始時から操作の順に付けられる。区分には、「src」と「dst」があり、利用者が直接指定したファイルは「src」となり、関連して変更が生じたファイルが「dst」となる。履歴の例を以下に示す。



ここで、「3.2.4 Options」の全履歴の表示(Show History All)メニューにおいて、表示される履歴の形式を説明する。全履歴表示画面を以下に示す、



全履歴の表示形式は、チェックボタンにより、以下の項目から選択できる。

(1) シーケンス (Sequence)

各ファイルに対するリファクタリング操作の順にその操作名の系列を表示する。
矢印 (?) は操作の順番を表現する。

(2) コマンド別 (Command)

リファクタリング操作を操作別に表示する。

(3) ファイル別 (File)

リファクタリング操作をその対象ファイル別に表示する。

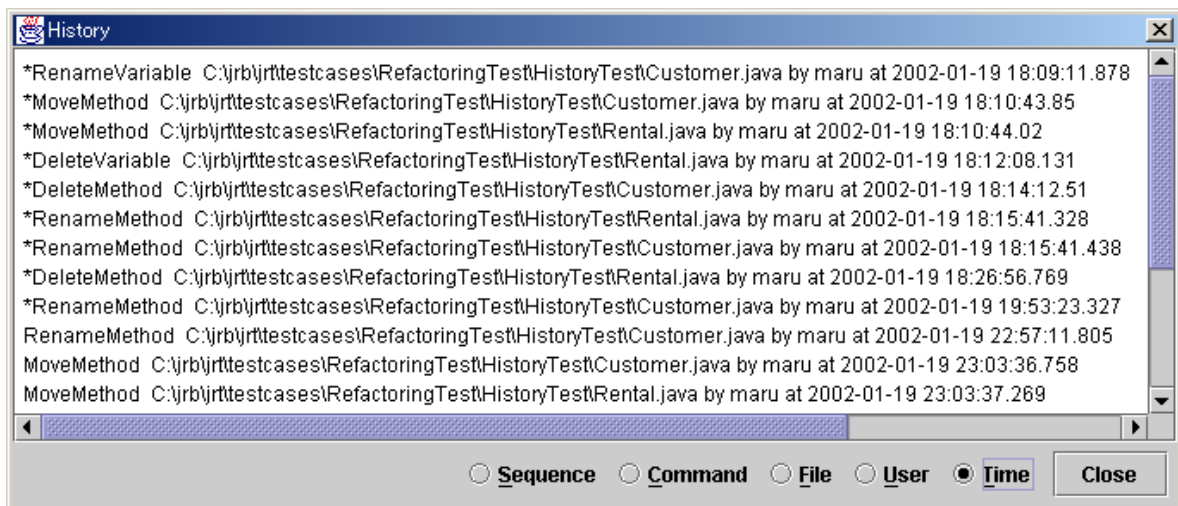
(4) 利用者別 (User)

リファクタリング操作をその操作を実行した利用者別に表示する。

(5) 時刻順 (Time)

リファクタリング操作をその実行時刻順に表示する。

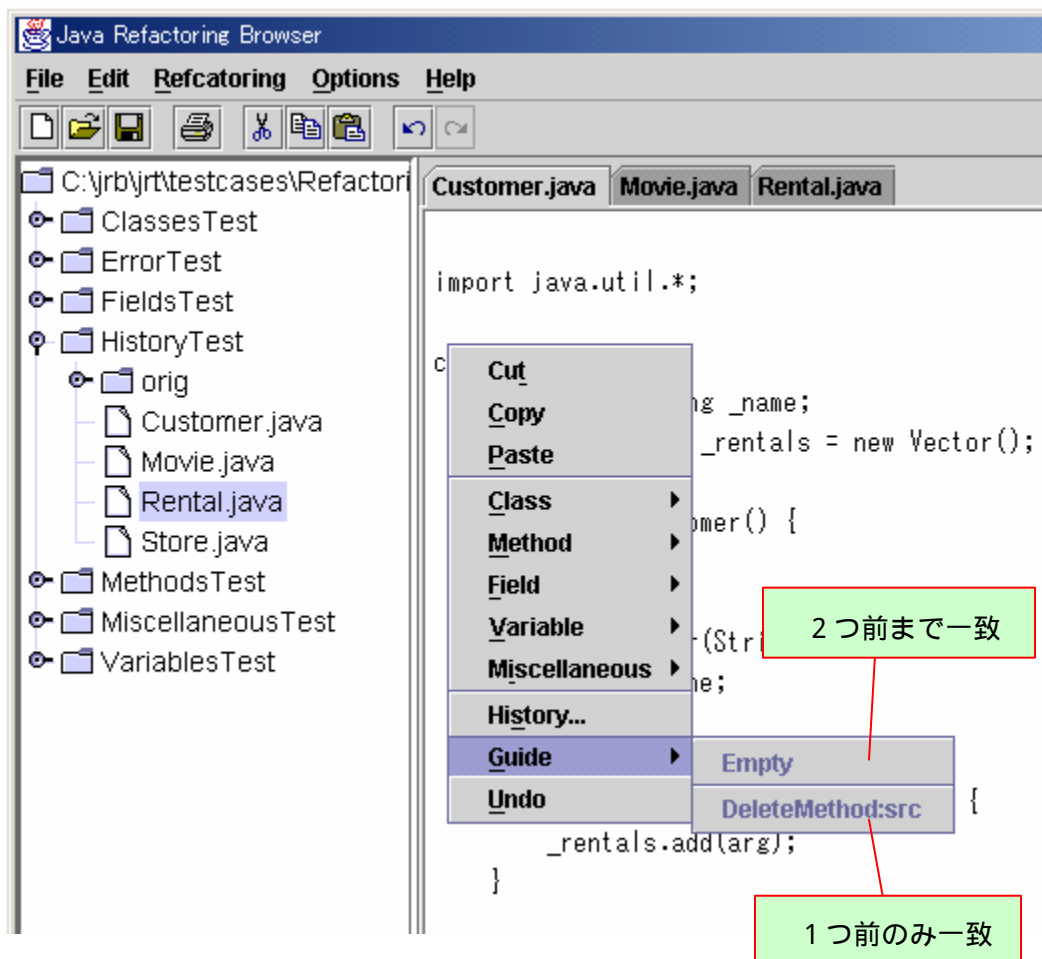
(2) ~ (5) の表示形式は , 「リファクタリング操作名」 + 「ファイル名」 + 「利用者名」 + 「実行時刻」である。時刻順に表示した例を以下に示す。



操作名の前に「 * 」が付加されている履歴は、既にファイルに登録されているものである。反対に、「 * 」が付いていないものは、メモリに存在する（ファイルに登録されていない）ものであり、JRB 終了時にファイルに記録される。

4.7 Guide メニュー（リファクタリング履歴の検索および次の操作の提案）

利用者の行った直前（1つあるいは2つ前）のリファクタリング操作をキーとして、過去の履歴から同一の操作を検索することで、次に行うべき操作を決定し、利用者に提案する。検索対象の履歴は、4.6 で説明した全履歴である。メニューの表示状態を以下に示す。



Guide メニューにより表示される項目は、2段に分かれている。上段は、2つ前まで一致（1つ前と2つまえの両方が一致）する場合に、過去において次にどのような操作が行われたのかを指す。下段は、1つ前のみ一致する場合に、次にどのような操作が行われたのかを指す。上図では、2つ前まで一致する操作は空であり、1つ前まで一致する操作が存在している。このメニューによると、現在の操作の次に、「メソッドの消去」操作を行うことを提案している。

一致する操作が複数存在する場合は、提案操作の過去における実行回数を調べ、それぞれの段で回数の多い順に5つを、メニュー項目の上から表示する。

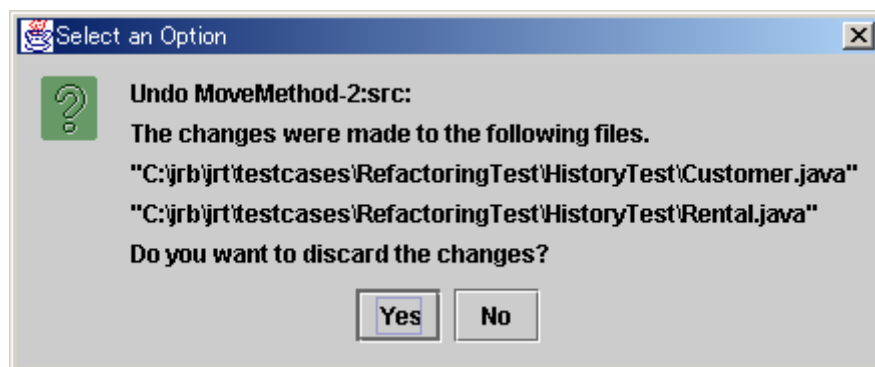
4.8 Undo メニュー（リファクタリング操作の取り消し）

このメニューでは、複数のファイルに対して同時に適用されたリファクタリング操作を一括して取り消すことが可能である。ただし、「3.2.2 Edit メニュー」の取り消しとの整合性をとるため、以下に示す制約を持つ。

- (a) 同時にリファクタリング操作が適用された全ファイルにおいて、その操作が取り消し履歴に存在し、かつ、最新であること。ここで、操作が取り消し履歴に存在するとは、Edit メニューの Undo で個別にその操作が取り消し可能であることを指す。ファイルの保存などにより、取り消し操作が不可能になった場合や既に Undo を実行した場合には、その操作は取り消し履歴から削除されている。また、操作が最新であるとは、その操作の後に別のリファクタリング操作が行われていない（文字の挿入や削除などの編集操作であれば良い）ことを指す。
- (b) 取り消すリファクタリング操作の後に行われた編集操作（文字の挿入や削除など）は、すべて破棄される。

このメニューにより取り消された操作は、やり直し履歴に蓄積されるため、Edit メニューの Redo でリファクタリングの取り消しをやり直すことも可能である。ただし、一括やり直しはできないため、各ファイルに対して個別に、Redo を実行する必要がある。

Undo 実行時の警告メニューを以下に示す。ここでは、「メソッドの移動」操作の取り消しを要求している。ダイアログによると、Customer.java ファイルと Rental.java ファイルへの変更が破棄されることがわかる。このダイアログに対して、Yes ボタンを押すと取り消しが実行される。取り消しの実行をキャンセルしたい場合は、No ボタンを押す。



5 . 開発ソフトウェアの障害対処方法

- (1) JRB が正常に起動しない場合，プロパティファイルの存在を確認すること．
- (2) 履歴が蓄積されない場合，JRB 実行ファイルが，インストールディレクトリに対して書き込み権限をもつかどうかを確認すること．
- (3) 実行環境によっては，立ち上がりや解析に長い時間がかかる．また，大量のファイルを解析した場合，Out of memory エラーがでることがある．よって，できるだけ高速の CPU と多くのメモリの使用を奨励する．
- (4) ソースコード解析中に GUI 操作を行うと，Java のエラーメッセージがでることがある．この場合，ソフトウェアの操作は継続実行可能であるが，編集画面上のソースコードと解析情報との間の整合性が保たれていない可能性があり，リファクタリングが正確に行われな
い．よって，編集中のソースコードをすべて保存し，再度 JRB を起動する必要がある．
- (5) JRB は Java 構文規則のすべてには対応していない．よって，リファクタリング対象ソースコードによっては，変換後のコードがコンパイルに失敗することがある．この場合は，通常のデバックを行うこと．
- (6) 全角文字を含むファイルを読み込んだ場合，文字コードによっては解析中に Java のエラーメッセージが出ることがある．特に，UNIX 実行環境では，読み込めたとしても編集画面上において正常に表示されない．全角文字を含むソースコードをリファクタリング対象とする際には，その文字コードをあらかじめ Unicode に変換しておくこと．

別添資料

なし